Федеральное государственное образовательное бюджетное учреждение  
высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ»**

|  |
| --- |
|  |
| Факультет информационных технологий и анализа больших данных |

Кафедра анализа данных и машинного обучения

Выпускная квалификационная работа

на тему: Разработка моделей и алгоритмов машинного обучения для прогнозирования рыночной стоимости жилой недвижимости

Направление подготовки:01.03.02 Прикладная математика и информатика

Профиль: Анализ данных и принятие решений в экономике и финансах

|  |  |  |  |
| --- | --- | --- | --- |
|  | Выполнил студент учебной группы | | |
|  | ПМ20-4 | | |
|  | Чернов Василий Александрович | подпись | |
|  | Научный руководитель работы | | |
|  | к.э.н., доцент | | |
|  | Абаев Владимир Александрович | | подпись |

|  |  |
| --- | --- |
|  | **ВКР соответствует предъявляемым**  **требованиям:**  Заведующий кафедрой анализа данных и машинного обучения, к.т.н., доцент |

\_\_\_\_\_\_\_\_\_\_\_\_ Д. А. Петросов

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Москва 2024

# Cодержание

[Cодержание 1](#_Toc166251316)

[ВВЕДЕНИЕ 2](#_Toc166251317)

[1.1. Актуальность темы 2](#_Toc166251318)

[1.1.1. Обоснование актуальности задачи прогнозирования стоимости жилья для покупателей, продавцов и инвесторов. 2](#_Toc166251319)

[1.1.2. Описание важности использования машинного обучения для улучшения точности прогнозов. 2](#_Toc166251320)

[1.2. Цель и задачи исследования 3](#_Toc166251321)

[1.2.1. Формулировка цели исследования 3](#_Toc166251322)

[1.2.2. Постановка конкретных задач, которые будут решены в работе. 3](#_Toc166251323)

[2. Теоретические основы 4](#_Toc166251324)

[2.1. Основы машинного обучения 4](#_Toc166251325)

[2.1.1. Определение основных терминов и понятий в области машинного обучения. 4](#_Toc166251326)

[2.1.2. Определение основных терминов и понятий в области глубокого обучения. 4](#_Toc166251327)

[2.2. Методы прогнозирования рыночной стоимости жилья 5](#_Toc166251328)

[2.2.1. Рассмотрение различных методов и подходов к прогнозированию стоимости жилья, включая классические статистические методы и методы машинного обучения. 5](#_Toc166251329)

[2.2.2. Анализ их преимуществ и ограничений. 22](#_Toc166251330)

[2.3. Предобработка данных 24](#_Toc166251331)

[2.3.1. Обзор этапов предобработки данных, включая сбор, очистку, преобразование и масштабирование. 24](#_Toc166251332)

[2.3.2. Рассмотрение методов обработки категориальных и числовых признаков. 26](#_Toc166251333)

[2.4. Выбор признаков 26](#_Toc166251334)

[2.4.1. Методы выбора наиболее информативных признаков для построения модели. 26](#_Toc166251335)

[2.4.2. Анализ корреляции и важности признаков. 27](#_Toc166251336)

[2.5. Метрики качества 27](#_Toc166251337)

[2.5.1. Описание популярных метрик оценки качества моделей регрессии (MAE, MSE, R-squared и др.). 27](#_Toc166251338)

[2.5.2. Какие метрики будут использоваться для оценки результатов в работе. 29](#_Toc166251339)

[3. Практическая часть 30](#_Toc166251340)

[3.1. Сбор и предобработка данных 30](#_Toc166251341)

[3.1.1. Описание источников данных о недвижимости 30](#_Toc166251342)

[3.1.2 Подробное описание процесса сбора данных и их предобработки 31](#_Toc166251343)

[3.2. Разработка и обучение моделей 42](#_Toc166251344)

[3.2.1. Выбор методов машинного обучения для задачи прогнозирования стоимости жилья 42](#_Toc166251345)

[3.2.2. Создание и обучение моделей на предобработанных данных 44](#_Toc166251346)

[3.2.3. Подбор гиперпараметров моделей 51](#_Toc166251347)

[3.3. Разработка глубокого обучения 55](#_Toc166251348)

[3.3.1. Выбор методов глубокого обучения для задачи прогнозирования стоимости жилья 55](#_Toc166251349)

[3.3.2 Создание и обучение нейросетей на предобработанных данных 56](#_Toc166251350)

[3.3.3. Подбор гиперпараметров моделей 58](#_Toc166251351)

[3.4. Оценка и сравнение моделей 59](#_Toc166251352)

[3.4.1. Использование выбранных метрик для оценки качества моделей 59](#_Toc166251353)

[3.4.2 Сравнительный анализ результатов разных моделей 67](#_Toc166251354)

[ЗАКЛЮЧЕНИЕ 69](#_Toc166251355)

[4.1. Сводка результатов 69](#_Toc166251356)

[4.1.1 Обобщение ключевых результатов исследования 69](#_Toc166251357)

[4.2. Практическая значимость 71](#_Toc166251358)

[4.2.1. Обсуждение практической значимости разработанных моделей для рынка недвижимости 71](#_Toc166251359)

[4.3. Ограничения и направления для будущих исследований 71](#_Toc166251360)

[4.3.1. Указания на ограничения проведенного исследования и предложение направлений для дальнейших исследований в данной области. 71](#_Toc166251361)

[5. СПИСОК ЛИТЕРАТУРЫ 73](#_Toc166251362)

[5.1. Указание на использованные в работе источники и литературу 73](#_Toc166251363)

[6. Приложение 77](#_Toc166251364)

[6.1. Код и скрипты, использованные для обработки данных и разработки моделей. 77](#_Toc166251365)

# ВВЕДЕНИЕ

## 1.1. Актуальность темы

## 1.1.1. Обоснование актуальности задачи прогнозирования стоимости жилья для покупателей, продавцов и инвесторов.

Современное машинное обучение и анализ данных стали важными инструментами для прогнозирования рыночной стоимости жилой недвижимости.

Исследование данной задачи является актуальным по нескольким причинам:

* Повышение эффективности рынка: Модели машинного обучения могут значительно улучшить точность и скорость оценки стоимости недвижимости, что важно как для инвесторов, так и для обычных покупателей и продавцов. Это помогает уменьшить асимметрию информации на рынке и делает процесс более прозрачным.
* Поддержка принятия решений: Автоматизированные инструменты оценки стоимости могут служить важным инструментом для поддержки решений о покупке, продаже и инвестировании в недвижимость, основываясь на объективных и актуализированных данных.
* Интеграция больших данных: На рынке недвижимости доступно большое количество данных, включая географическое расположение, характеристики объекта, исторические данные о сделках и даже социально-экономические индикаторы района. Модели машинного обучения могут обрабатывать эти большие объемы данных, выявляя сложные зависимости и тренды, которые могут ускользать от человеческого анализа.
* Адаптивность к изменениям рынка: Модели машинного обучения могут быстро адаптироваться к изменениям на рынке, обновляя оценки в реальном времени в ответ на новые данные и тренды.
* Развитие технологий и доступность: С развитием технологий и улучшением доступа к данным возрастает способность разрабатывать всё более точные и сложные модели машинного обучения, что также способствует их интеграции в различные бизнес-процессы в сфере недвижимости.
* Правовые и регуляторные аспекты: Развитие нормативной базы, регулирующей использование искусственного интеллекта и больших данных, также стимулирует применение этих технологий для улучшения прозрачности и справедливости рыночных оценок.

Таким образом, разработка и применение моделей машинного обучения для оценки стоимости жилой недвижимости не только отвечает потребностям современного рынка, но и способствует его дальнейшему развитию и оптимизации.

## 1.1.2. Описание важности использования машинного обучения для улучшения точности прогнозов.

Использование машинного обучения играет критическую роль в улучшении точности прогнозов из-за его способности обрабатывать большие объемы данных, выявлять сложные зависимости и паттерны, а также адаптироваться к изменяющимся условиям. Алгоритмы машинного обучения могут автоматически извлекать информацию из данных и использовать ее для создания моделей, способных делать точные прогнозы и предсказания.

## 1.2. Цель и задачи исследования

## 1.2.1. Формулировка цели исследования

Целью данной работы являются улучшение точности прогнозов стоимости недвижимости, посредствам создания моделей, способных с высокой точностью оценивать рыночную стоимость объектов на основе разнообразных данных..

## 1.2.2. Постановка конкретных задач, которые будут решены в работе.

Этот исследовательский проект создан для решения следующих задач:

* Сбор и анализ данных: Определение и сбор необходимых данных, включая исторические данные о сделках, характеристики объектов, демографические и экономические данные регионов.
* Обработка и очистка данных: Подготовка данных для анализа, включая их очистку, нормализацию и возможное преобразование.
* Выбор и настройка моделей машинного обучения: Исследование и выбор подходящих алгоритмов машинного обучения для обработки данных и предсказания стоимости.
* Разработка и валидация модели: Создание и тестирование модели на различных наборах данных для оценки её точности и устойчивости.

Эта цель и задачи направлены на создание эффективных и надежных инструментов для улучшения функционирования рынка жилой недвижимости через применение современных технологий машинного обучения.

# 2. Теоретические основы

## 2.1. Основы машинного обучения

## 2.1.1. Определение основных терминов и понятий в области машинного обучения.

Машинное обучение — это область искусственного интеллекта, которая занимается разработкой алгоритмов и моделей, способных извлекать закономерности и обучаться на основе данных, что позволяет делать прогнозы и принимать решения без явного программирования (Samuel, Arthur L., 1959) [1]. В контексте прогнозирования стоимости недвижимости, машинное обучение используется для создания моделей, которые могут анализировать характеристики домов и ценовые данные для предсказания стоимости (Краснопольский, Борис Л., 2017) [2].

## 2.1.2. Определение основных терминов и понятий в области глубокого обучения.

Глубокое обучение — это раздел машинного обучения, который использует нейронные сети с несколькими слоями для извлечения иерархии признаков из данных (LeCun, Yann, et al., 2015) [3]. В контексте оценки рыночной стоимости жилья, глубокое обучение может быть использовано для создания модели, способной предсказывать цену недвижимости на основе различных факторов (Иванов, Александр В., 2019) [4].

## 2.2. Методы прогнозирования рыночной стоимости жилья

## 2.2.1. Рассмотрение различных методов и подходов к прогнозированию стоимости жилья, включая классические статистические методы и методы машинного обучения.

**Линейная Регрессия**

Основная идея:

Линейная регрессия — это один из наиболее простых и широко используемых методов в задачах регрессии (Montgomery, Douglas C., et al., 2012) [5]. Она основана на предположении, что существует линейная зависимость между целевой переменной и набором признаков. Цель линейной регрессии — найти наилучшую линейную комбинацию признаков, которая наилучшим образом соответствует целевой переменной (Бахвалов, Павел А., 2016) [6].

Модель Линейной Регрессии:

Предположим, у нас есть n наблюдений и p признаков. Модель линейной регрессии может быть представлена следующим образом:

(1)

где:

- целевая переменная для i-го наблюдения.

- свободный член (пересечение) модели.

- коэффициенты регрессии, которые представляют собой параметры модели, которые нужно настроить.

- значения признаков для i-го наблюдения.

- случайная ошибка, которая представляет собой шум или невязку в модели. Она считается нормально распределенной с нулевым средним и постоянной дисперсией (

Задача Линейной Регрессии:

Цель линейной регрессии - оценить значения коэффициентов , чтобы минимизировать сумму квадратов ошибок (сумму квадратов невязок) между фактическими значениями целевой переменной () и предсказанными значениями ():

(2)

Часто минимизируется функционал средней квадратичной ошибки (Mean Squared Error, MSE):

(3)

- наблюдаемое значение целевой переменной для i-го наблюдения.

- предсказанное значение для i-го наблюдения.​

где n - количество наблюдений.

Решение:

Решение задачи линейной регрессии может быть найдено аналитически с использованием метода наименьших квадратов (Ordinary Least Squares, OLS). Результатом являются оценки коэффициентов , которые минимизируют MSE.

Формула для оценки коэффициентов:

(4)

​где:

- оценки коэффициентов.

X - матрица признаков, где каждая строка представляет собой наблюдение, а каждый столбец - признак.

y - вектор целевых переменных.

Прогноз:

После оценки коэффициентов модели, мы можем использовать их для предсказания значений целевой переменной для новых данных:

(5)

где - предсказанные значения целевой переменной для новых данных. (Bishop, Christopher M., 2006) [7].

**Ridge Regression**

Основная идея:

Ridge регрессия представляет собой расширение линейной регрессии, в которой к функции потерь добавляется регуляризационный член, чтобы предотвратить переобучение модели. Этот регуляризационный член штрафует большие значения коэффициентов модели (Hoerl, Arthur E., и Kennard, Robert W., 1970) [8].

Формула для Ridge регрессии:

Модель Ridge регрессии минимизирует следующую функцию потерь:

(6)

n - количество наблюдений в обучающей выборке.

p - количество признаков (параметров).​

- наблюдаемое значение целевой переменной для i-го наблюдения.

- предсказанное значение для i-го наблюдения.​

- коэффициенты модели для j-го признака.

- параметр регуляризации, который контролирует силу штрафа за большие значения коэффициентов. Большие значения α приводят к более сжатым (ближе к нулю) значениям коэффициентов.

**Lasso Regression**

Основная идея:

Lasso (Least Absolute Shrinkage and Selection Operator) регрессия также представляет собой расширение линейной регрессии с добавлением регуляризации. Однако в отличие от Ridge, Lasso использует L1-регуляризацию, которая склонна приводить к разреженным моделям, устанавливая некоторые коэффициенты в ноль и тем самым выполняя автоматический отбор признаков (Tibshirani, Robert, 1996) [9].

Формула для Lasso регрессии:

Модель Lasso регрессии минимизирует следующую функцию потерь:

(7)

где параметры такие же, как и в Ridge регрессии, а - также параметр регуляризации, но в данном случае он контролирует амплитуду регуляризации и влияет на то, какие коэффициенты будут установлены в ноль.

**Elastic Net Regression**

Основная идея:

Elastic Net регрессия является комбинацией Ridge и Lasso регрессий. Она включает как L1-регуляризации, так и L2-регуляризации и позволяет балансировать их влияние с помощью параметра (Zou, Hui и Hastie, Trevor, 2005) [10].

Формула для Elastic Net регрессии:

Модель Elastic Net регрессии минимизирует следующую функцию потерь:

(8)

где параметры такие же, как и в Ridge и Lasso регрессиях, а - это параметр, который контролирует баланс между L1- и L2-регуляризацией. Если = 0, то это эквивалентно Ridge регрессии, а если = 1, то это эквивалентно Lasso регрессии.

**Случайный лес**

Метод случайного леса (Random Forest) - это ансамблевый метод машинного обучения, который использует множество деревьев решений для выполнения задач классификации и регрессии.

Основная идея случайного леса заключается в том, что каждое дерево строится независимо на основе подмножества случайно выбранных признаков и случайной выборки наблюдений. Затем, принятие решения для нового объекта происходит путем голосования деревьев - каждое дерево выдает свой прогноз, и в качестве ответа выбирается наиболее популярный ответ (Брейман, 2001) [11].

Рассмотрим пример решения случайного леса для задачи регрессии:

Для каждого n = 1, … , N:

* Сгенерировать выборку с помощью бутстрэпа;
* Построить решающее дерево по выборке :

— при каждом разбиении сначала выбирается m случайных признаков из n исходных, и оптимальное разделение выборки ищется только среди них. Рекомендуется в задачах регрессии брать — m = n/3, где n — число признаков (Шмидт, 2018) [12].

— дерево строится, пока в каждом листе не более объектов или пока не достигнем определенной высоты дерева

Как строится дерево решения для задачи регрессии:

* При каждом разбиении считается прогнозное значение равное среднему всех эндогенных переменных в выборке и среднеквадратичную ошибку

(2) (9)

* Считаем прирост информации

(), (10) где – MSE изначальной выборки

и – MSE правого и левого разбиения

и – размер правого и левого разбиения

* Выбираем разбиение с наилучшим приростом информации
* Альтернативой IG является деление ориентируясь на сумму квадратов ошибок прогнозов

(2) (11)

* Необходимо высчитать RSS до деления и после деления, если сумма RSS правого и левого разбиения падает сильнее всего, то мы выбираем это разбиения для дальнейшего построения дерева
* Продолжаем дерево в направлении того листа, где MSE не равно нулю или меньше, чем в соседнем листе

**Бустинг**

AdaBoosting

Алгоритм AdaBoost для задач регрессии имеет сходную структуру с алгоритмом для задач классификации. Алгоритм начинается с случайного набора весов для каждого элемента в обучающей выборке. Затем на каждой итерации строится новый слабый регрессор, который пытается минимизировать сумму взвешенных абсолютных ошибок (взвешенная абсолютная ошибка). Эта ошибка определяется как сумма модулей разниц между предсказанным и правильным ответом, умноженная на вес каждого обучающего примера (Фройнд, Йоав; Шапире, Роберт, 1997) [13]. Обученный регрессор добавляется к ансамблю, а веса каждого обучающего примера пересчитываются таким образом, чтобы более важные примеры имели более высокие веса (Хасти, Тревор и др., 2009) [14].

Алгоритм AdaBoost для задач регрессии работает похожим образом, как и для задач классификации.

Шаг 1: Инициализация весов

Каждый обучающий пример i инициализируется весом w = 1, где n - количество обучающих примеров.

Шаг 2: Обучение базового регрессора

На каждой итерации алгоритма обучается слабый регрессор, который пытается минимизировать сумму взвешенных абсолютных ошибок:

(12)

где - правильный ответ для обучающего примера i, - предсказание регрессора для обучающего примера i, а - вес обучающего примера i.

Шаг 3: Расчет важности регрессора

После обучения регрессора рассчитывается его важность :

(13)

где – взвешенная ошибка регрессора, определяемая следующим образом:

(14)

где - предсказание t-го регрессора для обучающего примера i.

Важность регрессора зависит от его точности на обучающей выборке. Если регрессор ошибается часто, то его вес будет близок к нулю. Если регрессор точен, то его вес будет близок к бесконечности.

Шаг 4: Обновление весов

После расчета важности регрессора, веса обучающих примеров пересчитываются следующим образом:

(15)

где - правильный ответ для обучающего примера i, - предсказание t-го регрессора для обучающего примера i, а - важность t-го регрессора.

Это обновление весов делает более важные обучающие примеры более весомыми, если они были неправильно классифицированы на предыдущей итерации.

Шаг 5: Нормализация весов

После обновления весов они нормализуются, чтобы их сумма равнялась 1:

(16)

Это делается для того, чтобы на следующей итерации взвешенная ошибка была правильно рассчитана.

Шаг 6: Обучение следующего регрессора

Далее, обучается следующий регрессор, учитывая новые веса обучающих примеров. Обучение проводится на тех же данных, но веса для каждого обучающего примера учитываются с учетом ошибок предыдущих регрессоров.

Шаг 7: Общий ответ

После того, как все регрессоры обучены и их веса вычислены, ответ ансамбля регрессоров на новом примере вычисляется следующим образом:

(17)

где T - количество обученных регрессоров.

Общий ответ - это сумма ответов каждого регрессора, взвешенных на его важность. Этот ответ может использоваться для решения задач регрессии.

Шаг 8: Выбор количества регрессоров

Количество регрессоров в ансамбле является гиперпараметром, который необходимо определить. Обычно количество регрессоров выбирается с помощью кросс-валидации или других методов выбора модели.

В итоге, AdaBoost для задач регрессии используется для создания ансамбля слабых регрессоров, которые по-отдельности могут давать неточные ответы, но вместе образуют сильный алгоритм, который может решать сложные задачи регрессии.

**Градиентный бустинг**

Как и AdaBoost, градиентный бустинг является методом построения ансамбля из слабых моделей, однако использует другой подход к обучению. Он работает следующим образом:

Шаг 1: Инициализация

Градиентный бустинг начинается с инициализации ансамбля моделей. Обычно это делается путем создания простой модели, которая предсказывает среднее значение целевой переменной для всех обучающих примеров. Изначально все веса объектов равны 1/n, где n - количество объектов в обучающей выборке (Хейст, Гарет, и др., 2013) [15].

(18)

Шаг 2: Расчет остатков

Далее, для каждого обучающего примера рассчитывается остаток между его истинным значением и предсказанным значением модели (Брейман, Лео, 1996) [16].

(19)

Шаг 3: Обучение первой модели

Далее, обучается первая модель, которая должна предсказывать остатки, вычисленные на предыдущем шаге.

(20)

где L - функция потерь, которая описывает разницу между истинным значением и предсказанным значением, а h - функция слабой модели, которую мы пытаемся обучить.

Шаг 4: Обновление ансамбля

После того, как первая модель обучена, она добавляется в ансамбль моделей, взвешенный с помощью коэффициента .

(21)

Шаг 5: Расчет остатков для следующей модели

Далее, для каждого обучающего примера вычисляются остатки между его истинным значением и предсказанным значением ансамбля.

(22)

Шаг 6: Обучение следующей модели

Обучается следующая модель, которая должна предсказывать остатки, вычисленные на предыдущем шаге.

(23)

Шаг 7: Обновление ансамбля

После обучения следующей модели, она также добавляется в ансамбль моделей, взвешенный с помощью коэффициента .

(24)

Шаг 8: Повторение процесса

Шаги 5-7 повторяются до тех пор, пока не будет достигнуто определенное количество итераций или пока не будет достигнуто заданное условие остановки. Каждый раз, когда добавляется новая модель, ансамбль становится все более точным в предсказании целевой переменной.

Шаг 9: Получение прогноза

После того, как ансамбль моделей был обучен, он может быть использован для предсказания целевой переменной для новых, ранее не виденных данных. Для этого просто нужно применить ансамбль моделей к признаковому описанию каждого нового объекта.

(25)

где T - общее количество моделей в ансамбле, - коэффициент, с которым модель была добавлена в ансамбль, и - предсказанное значение целевой переменной для нового объекта.

Важно отметить, что градиентный бустинг не использует веса объектов, как это делает AdaBoost. Вместо этого градиентный бустинг рассчитывает остатки и обучает модели на этих остатках. Таким образом, каждая новая модель добавляет дополнительную информацию об ошибках предыдущих моделей, что помогает улучшить предсказания.

**Бэггинг**

Бэггинг (Bootstrap aggregating) - это метод построения ансамбля моделей, основанный на бутстрэп-выборке исходных данных и применении к каждой выборке отдельной модели, после чего результаты прогнозирования усредняются (Брейман, 1996) [16]

Подробно процесс бэггинга можно описать следующим образом:

1. Случайным образом извлекаем N выборок размера n с возвращением из исходного набора данных.
2. На каждой выборке обучаем модель .
3. Получаем итоговый прогноз на объекте x путем усреднения прогнозов от каждой модели:

(26)

Таким образом, бэггинг позволяет уменьшить дисперсию модели путем усреднения нескольких моделей, обученных на разных подвыборках исходных данных. Кроме того, использование бутстрэп-выборки позволяет улучшить устойчивость модели к выбросам и шуму в данных.

Важно отметить, что бэггинг можно применять к любым моделям, не только к деревьям решений, хотя именно в комбинации с деревьями бэггинг часто используется в практических приложениях.

**Стекинг**

Стекинг (Stacking) - это метод построения ансамбля моделей, в котором модели первого уровня (base models) обучаются на исходных данных, а затем их прогнозы используются как входные данные для обучения более высокоуровневой модели, называемой метамоделью (meta model) (Вольперт, 1992) [17]; (Брейман, 1996) [18]; (Зюко, 2003) [19]; (Ткачев, 2018) [20].

Подробно процесс стекинга можно описать следующим образом:

1. Разбиваем исходные данные на две части: обучающую и контрольную.
2. Обучаем модели первого уровня на обучающей выборке. Модели первого уровня могут быть любыми моделями машинного обучения, например, линейной регрессией, решающими деревьями, SVM, нейронными сетями и т.д. Обозначим модели первого уровня как , , ..., .
3. Получаем прогнозы моделей первого уровня на контрольной выборке. Обозначим прогнозы моделей первого уровня для i-го объекта контрольной выборки как , , ..., .
4. Строим новую матрицу признаков, где каждый объект представлен прогнозами моделей первого уровня. Новая матрица признаков будет иметь размерность (n,m), где n - количество объектов в контрольной выборке, а m - количество моделей первого уровня. Каждый объект i будет представлен в новой матрице признаков в виде вектора, состоящего из прогнозов моделей первого уровня: [, , ..., ]. Обозначим новую матрицу признаков как H.
5. Обучаем метамодель (meta model) на новой матрице признаков и целевых значениях из контрольной выборки.

Метамодель может быть любой моделью машинного обучения, например, линейной регрессией, решающим деревом, нейронной сетью и т.д. Обозначим метамодель как g(x).

Метамодель обучается по формуле:

g(x) = f(H(x)) (27)

где x - вектор признаков объекта из контрольной выборки, H(x) - вектор прогнозов моделей первого уровня для объекта x, а f(H(x)) - функция, которая принимает на вход вектор прогнозов моделей первого уровня и возвращает предсказание метамодели.

Чтобы обучить метамодель, нужно минимизировать функционал ошибки на контрольной выборке:

(28)

где n - количество объектов в контрольной выборке, - истинное значение целевой переменной для i-го объекта контрольной выборки, - предсказание метамодели для i-го объекта контрольной выборки, а - функция потерь.

1. Применяем стекинг к тестовым данным.

Для каждого объекта из тестовой выборки делаем прогнозы моделей первого уровня. Получаем новую матрицу признаков для тестовой выборки и применяем обученную метамодель для получения итоговых предсказаний.

Преимущества стекинга заключаются в том, что он позволяет использовать все преимущества разных моделей первого уровня, включая их специфические особенности и умения на различных типах данных. Однако, стекинг может быть сложен в настройке и требовать большего количества данных для обучения и оценки моделей.

**Регрессор голосования**

VotingRegressor - это ансамблевая модель, которая использует несколько моделей для предсказания целевой переменной. Каждая модель вносит свой вклад в итоговый прогноз, который вычисляется путем голосования или усреднения прогнозов Боровский, Алексей, 2021) [21].

1. Определяем модели первого уровня.
2. Для создания модели VotingRegressor необходимо определить набор базовых моделей. Это могут быть любые регрессионные модели, например, линейная регрессия, деревья решений, случайный лес и т.д. (Smith, John, 2019) [22].
3. Обучаем модели первого уровня на обучающей выборке.
4. Для каждой модели из набора обучаем модель на обучающей выборке. Обучение происходит путем минимизации функции потерь, например, среднеквадратичной ошибки.
5. Получаем прогнозы от моделей первого уровня (Жуков, Павел, 2020) [23].
6. Для каждого объекта из тестовой выборки получаем прогнозы от каждой модели первого уровня. Полученные значения используются как признаки для модели второго уровня.
7. Обучаем модель второго уровня на прогнозах моделей первого уровня.
8. Для создания итогового прогноза используется модель второго уровня, которая обучается на прогнозах, полученных от моделей первого уровня. Обычно это может быть линейная регрессия или другая модель, которая может объединить прогнозы различных моделей первого уровня. Итоговый прогноз рассчитывается путем голосования или усреднения прогнозов моделей первого уровня.
9. Применяем модель к тестовым данным.
10. Для каждого объекта из тестовой выборки делаем прогнозы моделей первого уровня. Получаем новую матрицу признаков для тестовой выборки и применяем обученную модель второго уровня для получения итоговых предсказаний.

Преимущества VotingRegressor заключаются в том, что он позволяет объединить прогнозы различных моделей, что может привести к более точным прогнозам, чем отдельные модели. Однако, VotingRegressor может быть менее интерпретируемым, чем отдельные модели, и может потребовать большего количества ресурсов для обучения.

**XGBoost**

XGBoost (eXtreme Gradient Boosting) — это эффективная реализация алгоритма градиентного усиления для регрессии и классификации. Основные моменты XGBoost с формулами:

Обучение модели

Обучение XGBoost основывается на аддитивной стратегии, где новые модели вставляются для исправления ошибок предыдущих моделей (Чен, Тяньци, и Гю, Карлос, 2016) [24]. В каждой итерации добавляется дерево, которое минимизирует функцию потерь:

(29)

где:

L — функция потерь;

— параметры модели;

— дифференцируемая выпуклая функция потерь;

— предсказание на i-й точке данных на t-й итерации;

— регуляризационный терм, управляющий сложностью модели;

— k-е дерево решений.

Функция потерь

Функция потерь может быть, например, среднеквадратичной ошибкой для регрессии:

(30)

Регуляризация

Регуляризационный терм помогает управлять сложностью модели:

(31)

где:

— количество конечных узлов в дереве;

— вектор значений в конечных узлах;

и — параметры регуляризации.

Оптимизация

XGBoost использует вторую производную функции потерь для более эффективного нахождения направления для оптимизации. Обновление значения в каждом листе дерева зависит от градиента g и гессиана h функции потерь:

(32)

Важность признаков

XGBoost может оценить важность признаков, позволяя понять, какие признаки вносят наибольший вклад в предсказания. Это может быть основано на общем уменьшении функции потерь благодаря признаку или на частоте использования признака в разбиениях.

Настройка гиперпараметров

Эффективность XGBoost часто зависит от настройки гиперпараметров, таких как: (Кузнецов, Сергей, 2018) [25].

- learning\_rate: шаг обучения;

- max\_depth: максимальная глубина деревьев;

- subsample: доля данных для обучения каждого дерева;

- colsample\_bytree: доля признаков для обучения каждого дерева;

- n\_estimators: количество деревьев в ансамбле.

XGBoost славится своей производительностью и точностью, а также способностью работать на больших наборах данных с высокой скоростью. Это делает его популярным выбором во многих прикладных задачах машинного обучения.

**LightGBM**

LightGBM (Light Gradient Boosting Machine) — это градиентный бустинговый фреймворк, который использует алгоритмы обучения на основе деревьев решений. Он оптимизирован для больших объёмов данных и высокой эффективности. Вот ключевые аспекты LightGBM с формулами:

Обучение модели

LightGBM также использует аддитивную стратегию, подобно XGBoost, где новые деревья создаются для корректировки ошибок предыдущих:

(33)

где:

— функция потерь;

— выпуклая функция потерь, например, логистическая потеря для классификации;

— предсказание на предыдущей итерации;

— дерево, добавляемое на итерации t;

— регуляризация, учитывающая сложность дерева.

Функция потерь

Для регрессии LightGBM может использовать среднеквадратичное отклонение (MSE) как функцию потерь:

(34)

Регуляризация

Регуляризация дерева решений в LightGBM может включать как L1 (LASSO), так и L2 (Ridge) регуляризацию для весов:

(35)

где — значение в листовом узле дерева, T — общее количество листов в дереве.

Особенности алгоритма

LightGBM отличается от других алгоритмов градиентного бустинга двумя основными способами:

- GOSS (Gradient-based One-Side Sampling):\*\* LightGBM пропускает выборку данных с меньшими градиентами и использует только те с большими градиентами, что ускоряет обучение, сокращая количество данных, необходимых для вычислений.

- EFB (Exclusive Feature Bundling):\*\* Автоматическое объединение исключающих друг друга (разреженных) признаков в один, что уменьшает количество признаков и ускоряет обучение.

Настройка гиперпараметров

Настройка гиперпараметров в LightGBM может включать параметры, аналогичные XGBoost, такие как `learning\_rate`, `num\_leaves` (количество листьев в деревьях), `feature\_fraction` (используется для подвыборки признаков), `bagging\_fraction` и `bagging\_freq` (используются для подвыборки данных), что позволяет настроить сложность модели и предотвратить переобучение.

LightGBM особенно полезен при работе с большими наборами данных и обладает высокой скоростью обучения при сохранении высокой точности моделирования (Прохоров, Александр, 2019) [26].

**Нейронный сети**

Многослойный персептрон (MLP) представляет собой классическую архитектуру нейронной сети, состоящую из нескольких слоев: входного слоя, одного или нескольких скрытых слоев и выходного слоя. Каждый нейрон в одном слое связан с каждым нейроном в следующем слое. Вот как формально определить MLP:

1. Входной слой: Первый слой нейронов, который принимает входные данные. Каждый нейрон в этом слое представляет один из признаков входных данных.
2. Скрытые слои: Один или несколько слоев, которые следуют за входным слоем и содержат скрытые нейроны. Эти нейроны вычисляют нелинейные комбинации входных данных и передают их на следующие слои.
3. Выходной слой: Последний слой, который выдает результат работы нейронной сети. Количество нейронов в этом слое зависит от задачи, например, для задачи регрессии может быть один нейрон, а для задачи классификации - несколько нейронов (по числу классов) (Heaton, Jeff, 2017) [27]

Каждый нейрон в MLP имеет веса, которые определяют его вклад в выход нейрона, а также смещение (bias), которое учитывается при вычислении активации нейрона. Для каждого нейрона вычисляется взвешенная сумма входов, которая затем подается на функцию активации для получения выхода. Обычно в качестве функции активации используется нелинейная функция, такая как сигмоидальная функция или функция ReLU (Rectified Linear Unit) (Hornik, Kurt, 1991) [28].

Формально, для одного нейрона j в слое l вычисление активации можно записать следующим образом:

(36)

где:

- взвешенная сумма входов для нейрона j в слое l

- вес связи между нейроном i в слое (l−1) и нейроном j в слое l

- выход нейрона i в предыдущем слое

- смещение (bias) для нейрона j в слое l

- количество нейронов в предыдущем слое

После вычисления взвешенной суммы , активация нейрона j вычисляется с использованием функции активации σ:

(37)

Этот процесс повторяется для каждого слоя нейронной сети, начиная с входного слоя и заканчивая выходным слоем. Обучение MLP обычно включает в себя применение алгоритмов обратного распространения ошибки (backpropagation) для настройки весов и смещений с целью минимизации функции потерь.

## 2.2.2. Анализ их преимуществ и ограничений.

Рассмотрим преимущества и ограничения каждой из упомянутых моделей машинного обучения при их использовании для решения задачи регрессии:

**Linear (Линейная регрессия)**

* Преимущества: Простота реализации и интерпретации, требует мало вычислительных ресурсов.
* Ограничения: Предполагает линейную зависимость между переменными, плохо справляется с нелинейными зависимостями и выбросами.

**Lasso (Лассо регрессия)**

* Преимущества: Помогает в выборе признаков за счет штрафа за сложность модели, эффективно сокращает размерность.
* Ограничения: Может привести к чрезмерной регуляризации, что ухудшит предсказательную способность модели.

**ElasticNet**

* Преимущества: Комбинирует штрафы Lasso и Ridge, обеспечивая баланс между выбором признаков и регуляризацией.
* Ограничения: Настройка двух параметров регуляризации может быть сложной.

**Ridge (Гребневая регрессия)**

* Преимущества: Хорошо справляется с мультиколлинеарностью за счет добавления штрафа к квадратам коэффициентов.
* Ограничения: Не уменьшает количество признаков, все они остаются в модели.

**RandomForestRegressor**

* Преимущества: Хорошо справляется с нелинейностями и взаимодействиями между признаками, нечувствителен к масштабу данных.
* Ограничения: Тенденция к переобучению на небольших наборах данных, требует больших вычислительных ресурсов.

**AdaBoostRegressor**

* Преимущества: Улучшает предсказательную способность за счет последовательного уточнения слабых предсказаний.
* Ограничения: Чувствителен к шуму и выбросам в данных.

**GradientBoostingRegressor**

* Преимущества: Очень эффективен за счет последовательного уменьшения ошибок, часто дает высокую точность.
* Ограничения: Может быть склонен к переобучению, требует тщательной настройки параметров.

**BaggingRegressor**

* Преимущества: Уменьшает переобучение, стабилизирует меньше подверженные шуму предсказания.
* Ограничения: Менее эффективен, когда модели слабо коррелированы.

**StackingRegressor**

* Преимущества: Комбинирует предсказания различных моделей, потенциально улучшая общую точность.
* Ограничения: Высокая сложность модели, может быть сложно интерпретировать.

**VotingRegressor**

* Преимущества: Объединяет разные модели для уменьшения дисперсии и улучшения стабильности.
* Ограничения: Требуется настройка весов голосов, может не дать значительного улучшения.

**XGBRegressor**

* Преимущества: Высокоэффективный и быстрый, хорошо масштабируется, поддерживает различные функции потерь.
* Ограничения: Может быть сложен в настройке, склонен к переобучению без должной регуляризации.

**LGBMRegressor**

* Преимущества: Быстрее и эффективнее XGB в больших наборах данных, эффективно обрабатывает большое количество признаков.
* Ограничения: Менее интерпретируемый по сравнению с деревьями меньшей глубины.

**NN (Нейронные сети)**

* Преимущества: Гибкость в моделировании сложных нелинейных взаимосвязей, хорошо справляется с большими и сложными данными.
* Ограничения: Требует больших вычислительных ресурсов, сложна в настройке и интерпретации, риск переобучения.

## 2.3. Предобработка данных

## 2.3.1. Обзор этапов предобработки данных, включая сбор, очистку, преобразование и масштабирование.

1. Удаление ненужных столбцов:

Столбцы удаляются, так как они не представляют интереса для анализа рыночной стоимости жилья.

1. Обработка пропущенных значений:

Проверяется наличие пропущенных значений в каждом столбце, их количество выводится на экран. Если присутствуют пропущенные значения, необходимо принять решение об их заполнении или удалении.

1. Кодирование категориальных признаков:

Категориальные признаки, представленные в виде строковых значений, кодируются с помощью метода LabelEncoder для преобразования их в числовые значения. Это необходимо для того, чтобы модель машинного обучения могла работать с этими данными.

1. Анализ корреляции признаков:

С помощью тепловой карты корреляции проверяется взаимосвязь между признаками. Это помогает определить, какие признаки коррелируют с целевой переменной (ценой жилья), а также какие признаки коррелируют между собой.

Некоторые признаки могут быть удалены из набора данных, если они либо не коррелируют с целевой переменной, либо имеют высокую корреляцию с другими признаками.

1. Преобладание классов:

Проводится анализ данных для выявления и предобработки столбцов с преобладанием нулевых значений.

1. Визуализация данных:

Проводится визуальный анализ данных с помощью диаграмм рассеяния и ящиков с усами для каждого числового признака. Это помогает понять распределение данных и выявить какие-либо аномалии или необычные паттерны.

1. Обработка выбросов:

Выбросы в числовых признаках анализируются с помощью ящика с усами (boxplot). Выбросы определяются как значения, находящиеся за пределами 1.5 межквартильных расстояний от первого и третьего квартилей. Выбросы удаляются из набора данных.

1. Анализ важности признаков:

Выполняет обучение модели случайного леса для прогнозирования цены (price) на основе остальных признаков, содержащихся в данных. Такой подход позволяет обучить модель, которая учитывает важность различных признаков для предсказания цены и позволяет провести анализ важности признаков для лучшего понимания данных и улучшения модели.

Эти методы предобработки данных помогают подготовить данные для построения модели машинного обучения, которая может предсказывать рыночную стоимость жилья на основе имеющихся признаков.

## 2.3.2. Рассмотрение методов обработки категориальных и числовых признаков.

В процессе подготовки данных для машинного обучения часто возникает необходимость трансформации категориальных признаков в числовой формат, так как большинство алгоритмов машинного обучения лучше работают с числовыми данными. Один из распространённых методов — использование LabelEncoder, который предоставляет библиотека scikit-learn.

LabelEncoder преобразует категориальные значения в последовательные целочисленные коды. Каждому уникальному категориальному значению столбца присваивается уникальный индекс от 0 до N−1, где N — количество уникальных категорий. Например, для категорий ["красный", "синий", "зеленый"] LabelEncoder присвоит значения 0, 1 и 2 соответственно.

## 2.4. Выбор признаков

## 2.4.1. Методы выбора наиболее информативных признаков для построения модели.

В процессе построения моделей машинного обучения важную роль играет выбор наиболее информативных признаков, который может существенно повысить качество и эффективность предсказательной модели. Один из популярных методов выбора признаков — использование алгоритма RandomForest, который является ансамблевым методом, основанным на деревьях решений.

RandomForest состоит из большого числа деревьев решений, каждое из которых обучается на случайной подвыборке набора данных с использованием случайного подмножества признаков. Этот метод объединяет предсказания отдельных деревьев, что обычно приводит к улучшению общей точности и устойчивости модели по сравнению с одним деревом решений.

## 2.4.2. Анализ корреляции и важности признаков.

В контексте статистического анализа и машинного обучения, анализ корреляции играет ключевую роль в определении взаимосвязей между переменными. Матрица корреляций — это инструмент, который позволяет визуализировать и количественно оценить степень линейной зависимости между парами признаков в наборе данных. Это особенно полезно для выявления признаков, которые могут быть избыточными или чрезмерно взаимозависимыми, что может привести к проблемам мультиколлинеарности в регрессионных моделях.

## 2.5. Метрики качества

## 2.5.1. Описание популярных метрик оценки качества моделей регрессии (MAE, MSE, R-squared и др.).

В этой работе были использованы такие метрики как R2 , MAE, MSE, MAPE, RMSPE.

1. R^2 (коэффициент детерминации) - это мера соответствия модели данных. Он показывает, как хорошо модель описывает вариацию целевой перемен ной. Коэффициент детерминации принимает значения от 0 до 1, где 0 означает, что модель не объясняет вариацию целевой переменной, а 1 означает, что модель идеально соответствует данным.

Математически, R^2 рассчитывается как отношение объясненной дисперсии к общей дисперсии:

(38)

2  (39) 2 (40)

где RSS - сумма квадратов остатков (разница между фактическими значениями и прогнозами модели), TSS - общая сумма квадратов (разница между фактическими значениями и средним значением целевой переменной).

1. MAE (Mean Absolute Error) - это метрика, используемая для оценки точности прогнозных моделей. Она измеряет среднее абсолютное отклонение между прогнозами модели и соответствующими фактическими значениями.

Формула MAE выглядит следующим образом:

(41)

где:

n - количество наблюдений (размер выборки)

- прогнозные значения

- фактические значения

В формуле вычисляется абсолютное значение разницы между каждым прогнозом и соответствующим фактическим значением . Затем берется сумма всех этих абсолютных разниц, и результат делится на количество наблюдений n. Таким образом, MAE представляет среднее абсолютное отклонение между прогнозами и фактическими значениями.

1. Средняя квадратичная ошибка (Mean Squared Error, MSE) вычисляется по следующей формуле:

(42)

где:

n - количество наблюдений (размер выборки)

- прогнозные значения

- фактические значения

Эта метрика вычисляет среднее арифметическое квадратов разностей между фактическими и прогнозными значениями. MSE штрафует за большие ошибки сильнее, чем MAE. Чем меньше значение MSE, тем лучше качество модели.

1. Средняя абсолютная процентная ошибка (Mean Absolute Percentage Error, MAPE) вычисляется по формуле:

(43)

где:

n - количество наблюдений (размер выборки)

- прогнозные значения

- фактические значения

MAPE выражает среднее процентное отклонение между фактическими и прогнозными значениями. Эта метрика особенно полезна при оценке точности прогнозов в процентах. Чем меньше значение MAPE, тем лучше качество модели.

MAPE хорошо интерпретируется, так как показывает ошибку в процентах относительно фактических значений.

1. Среднеквадратичная процентная ошибка (Root Mean Squared Percentage Error, RMSPE) вычисляется похожим образом на среднюю абсолютную процентную ошибку (MAPE), но с использованием квадратов отклонений:

(44)

где:

n - количество наблюдений (размер выборки)

- прогнозные значения

- фактические значения

RMSPE измеряет среднеквадратичное отклонение в процентах между фактическими и прогнозными значениями. Эта метрика также часто используется для оценки точности прогнозов. Чем меньше значение RMSPE, тем лучше соответствие прогноза и фактических данных.

RMSPE помогает оценить точность модели с целью минимизации квадратических отклонений относительно фактических значений.

## 2.5.2. Какие метрики будут использоваться для оценки результатов в работе.

Когда речь идет о оценке рыночной стоимости жилья, все эти метрики могут быть полезны в зависимости от контекста. Например, R2 дает общую оценку качества модели, MAE и MSE измеряют среднюю ошибку модели в единицах измерения цены, а MAPE и RMSPE позволяют оценить точность модели в процентах относительно фактических значений (James, Gareth et al., 2013) [29]

# 3. Практическая часть

## 3.1. Сбор и предобработка данных

## 3.1.1. Описание источников данных о недвижимости

Датасет "House Sales in King County, USA" является обширным набором данных, который предоставляет подробную информацию о продажах домов в округе Кинг, штат Вашингтон, США, за период с мая 2014 года по май 2015 года. Этот датасет включает 21,613 записей и охватывает широкий спектр характеристик, которые могут быть использованы для анализа и прогнозирования цен на жилую недвижимость. Вот более подробное описание характеристик, доступных в этом датасете:

- id - Уникальный идентификатор для каждой продажи.

- date - Дата продажи, которая помогает анализировать сезонные тенденции и динамику рынка.

- price - Цена продажи, является целевой переменной для задач прогнозирования стоимости.

- bedrooms и bathrooms - Количество спален и ванных комнат, что важно для оценки размера и удобства дома.

- sqft\_living и sqft\_lot - Жилая площадь и общая площадь участка соответственно, измеренные в квадратных футах.

- floors - Количество этажей, что может влиять на стоимость и привлекательность дома.

- waterfront - Индикатор наличия выхода к водоему, что значительно может увеличивать стоимость недвижимости.

- view - Оценка вида из дома, например, на озера или горы, что также влияет на цену.

- condition и grade - Оценка состояния дома и его строительного качества, что используется для определения потребности в ремонте и улучшениях.

- sqft\_above и sqft\_basement - Площадь надземной и подземной частей дома, что позволяет оценить дополнительное пространство и его функциональность.

- yr\_built и yr\_renovated - Год постройки и последней реновации, что важно для оценки возраста дома и его современности.

- zipcode, lat, и long - Почтовый индекс и координаты местоположения, что полезно для географического анализа и определения стоимости по разным регионам.

- sqft\_living15 и sqft\_lot15 - Площади жилых помещений и участков для 15 ближайших домов, что может использоваться для сравнительного анализа соседних объектов.

Эти данные можно использовать для разработки сложных моделей машинного обучения и нейросетей для прогнозирования стоимости домов, анализа рыночных тенденций и изучения влияния различных факторов на стоимость недвижимости в округе Кинг.

## 3.1.2 Подробное описание процесса сбора данных и их предобработки

Данные были взяты из Kaggle. Это крупнейшее в мире сообщество специалистов в области обработки данных, обладающее мощными инструментами и ресурсами.

Для сбора данных о продажах жилой недвижимости обычно используются записи из государственных регистрационных баз, данные агентств недвижимости и информационных систем, которые собирают и агрегируют информацию о заключённых сделках. Это включает в себя следующие шаги:

* Источники данных: Данные могут поступать из множества источников, включая государственные реестры недвижимости, базы данных MLS (Multiple Listing Service), которыми управляют риэлторские ассоциации, а также из финансовых отчётов и баз данных кредитных организаций.
* Агрегация данных: Информация с разных источников обрабатывается и интегрируется в единую базу данных. Это включает в себя стандартизацию форматов данных, проверку на точность и удаление дубликатов.
* Анонимизация: Для того чтобы соблюдать приватность и законы о защите данных, личная информация о владельцах удаляется или анонимизируется перед публикацией данных.
* Обогащение данных: Данные могут быть обогащены дополнительной информацией, такой как демографические данные региона, экономические индикаторы или данные о школьных округах.
* Контроль качества: Перед публикацией проводится тщательная проверка данных на предмет ошибок, пропусков и непоследовательностей.

После этих процедур данные становятся доступны для использования исследователями, аналитиками и разработчиками, которые могут применять методы машинного обучения и нейросетей для анализа рынка недвижимости, прогнозирования цен и изучения рыночных тенденций.

Прежде чем приступить к прогнозированию рыночной стоимости жилой недвижимости, необходимо правильно выполнить предобработку данных. Это важный этап, который влияет на качество анализа и результаты моделирования.

  
  
 Рисунок 1. Загрузка данных

Эта команда загружает данные из файла CSV в DataFrame new\_data, который содержит данные о продажах домов. Каждая строка представляет собой отдельный продажный случай с различными атрибутами, такими как цена, количество спален, жилая площадь и тд.



Рисунок 2. Удаление ненужных столбцов

В этом шаге удаляются столбцы id и date, которые могут быть не нужны для последующего анализа. Удаление этих столбцов помогает сосредоточить анализ на более важных атрибутах.

Рисунок 3. Проверка размера данных

Эта команда выводит размерность данных (количество строк и столбцов). Это помогает понять, сколько наблюдений и переменных содержится в наборе данных после предварительной обработки.

 Рисунок 4. Статистическое описание данных

Метод describe() предоставляет статистический суммарный анализ для числовых переменных, включая среднее, стандартное отклонение, минимальное и максимальное значения, а также квартили. Это дает представление о распределении данных.

 Рисунок 5. Проверка типов данных

Эта команда выводит типы данных для каждого столбца в DataFrame. Это важно для определения, нужно ли конвертировать данные перед анализом, например, преобразовать строковые данные в числовые.

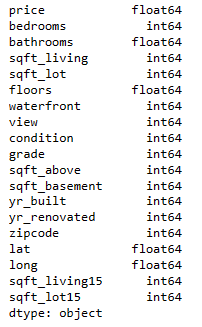


Рисунок 6. Типы данных

Здесь видно, что все типы данных являются числовыми, следовательно у нас нет необходимости кодировать категориальные признаки для того, чтобы модель машинного обучения могла работать с этими данными.

 Рисунок 7. Проверка на наличие пропущенных значений

Этот метод выявляет количество пропущенных значений в каждом столбце. Это критически важно, так как пропущенные данные могут существенно повлиять на результаты анализа и моделирования. В зависимости от результатов этого анализа можно решить, заполнять ли пропущенные значения, удалять строки или столбцы с пропусками или использовать другие методы обработки.

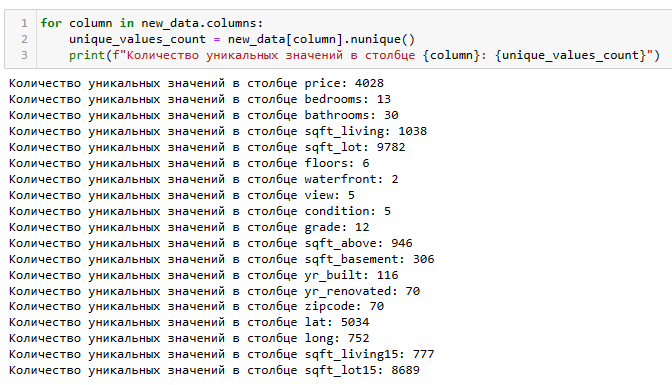


Рисунок 8. Подсчет уникальных значений

Этот шаг направлен на определение уникальности данных в каждом столбце. Код проходит по каждому столбцу датафрейма “new\_data”, вычисляя количество уникальных значений. Это необходимо для обнаружения аномалий, слишком мало или слишком много уникальных значений может указывать на ошибки в данных или неправильную предобработку.

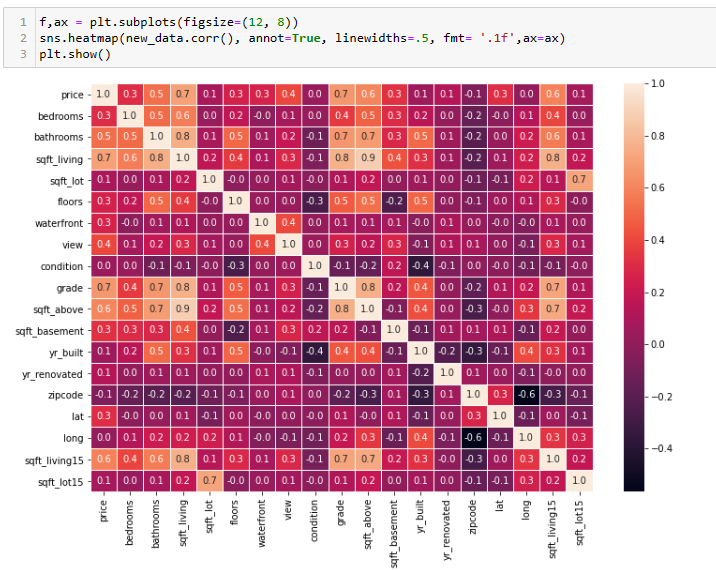


Рисунок 9. Визуализация корреляций

Построение тепловой карты корреляций между признаками помогает понять взаимосвязи между переменными. Для этого используется метод corr() для создания матрицы корреляций и библиотека Seaborn для её визуализации. Важность этого шага:

* Анализ взаимосвязей: Понимание, как признаки связаны друг с другом. Например, высокая положительная корреляция между размером дома и его стоимостью ожидаема.
* Мультиколлинеарность: Выявление признаков с высокой корреляцией, что может негативно сказаться на некоторых моделях машинного обучения.
* Подготовка к моделированию: Исключение или модификация признаков для повышения качества и устойчивости моделей.

Тепловая карта показывает степень корреляции цветом, где темные оттенки указывают на сильную связь (как положительную, так и отрицательную). Это визуальное представление делает анализ более наглядным и понятным.

 Рисунок 10. Удаление столбцов с высокой корреляцией

Шаги предобработки данных с удалением столбцов из-за высокой корреляции

1. Анализ корреляции данных

Перед удалением столбцов был проведен анализ корреляционной матрицы датасета. Эта матрица позволяет оценить степень линейной зависимости между переменными. На основе этой матрицы можно увидеть, какие признаки сильно коррелируют друг с другом. Высокая корреляция между переменными может привести к проблемам мультиколлинеарности в моделях машинного обучения, что затрудняет интерпретацию влияния отдельных признаков на зависимую переменную.

2. Решение об удалении столбцов

На основе корреляционной матрицы было принято решение удалить следующие столбцы:

sqft\_above – общая площадь жилых помещений, не включая подвал. Этот признак сильно коррелирует с sqft\_living, что может быть объяснено тем, что общая площадь дома в большинстве случаев включает площадь без подвала.

grade – оценка качества материалов и отделки, также показала высокую корреляцию с ценой и площадью жилья, что делает этот признак избыточным при наличии других, более специфичных метрик.

bathrooms – количество ванных комнат, которое часто коррелирует с общей площадью жилого пространства и количеством спален.

sqft\_living15 – площадь жилой зоны в радиусе 15 ближайших домов, этот показатель может дублировать информацию, содержащуюся в sqft\_living.

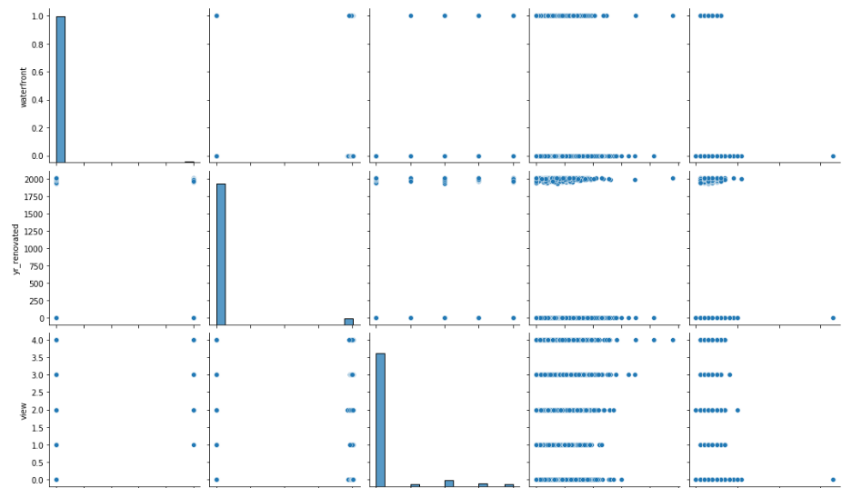
3. Удаление столбцов

Для удаления столбцов использовалась функция drop библиотеки pandas. Эта операция уменьшает размерность данных и упрощает модель, уменьшая риск переобучения.



Рисунок 11. Анализ данных с нулевыми значениями

Данный анализ позволил выявить признаки с наибольшим количеством нулевых значений, что может указывать на ограниченную полезность этих признаков для моделирования или их избыточность.



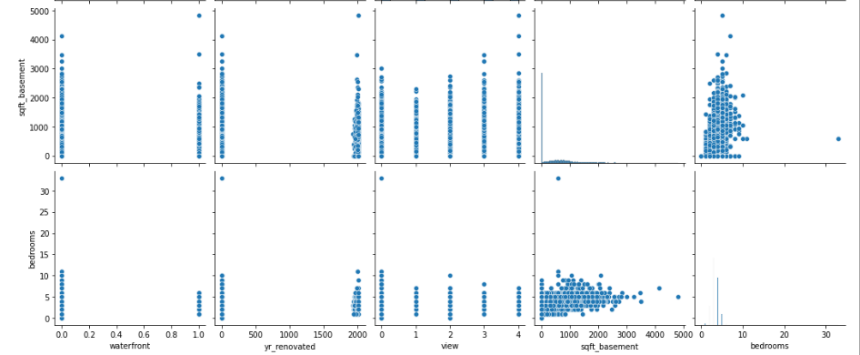


Рисунок 12. Парные графики признаков с нулевыми значениями

На основании полученных данных о проценте нулей был сформирован список столбцов с наибольшим количеством нулевых значений для дальнейшего визуального анализа с помощью парных графиков (pairplot из библиотеки seaborn). Это позволило визуализировать распределения и взаимосвязи между выбранными признаками, что важно для понимания структуры данных и выявления возможных аномалий или закономерностей.

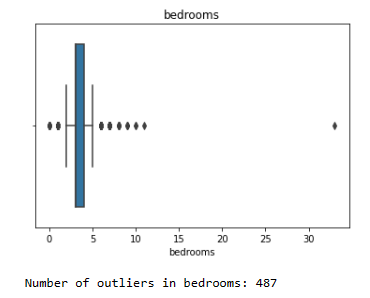
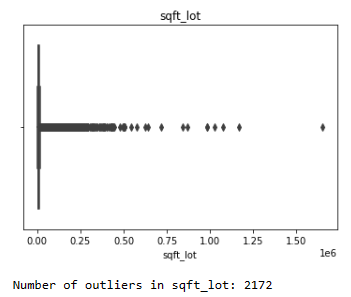


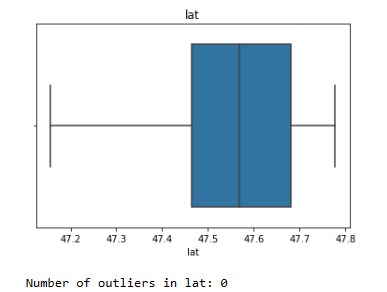
Рисунок 13. Удалении столбцов с нулевыми значениями

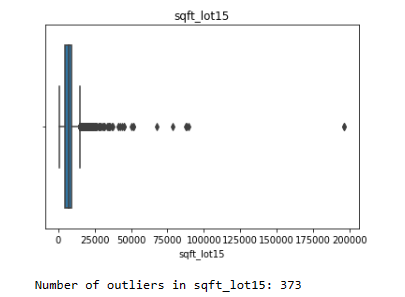
После анализа было решено удалить следующие столбцы из-за их высокого процента нулевых значений и потенциально ограниченной предсказательной способности:

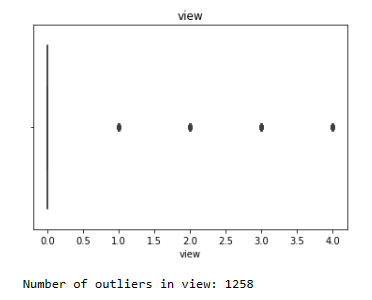
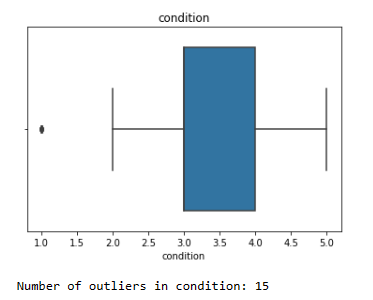
* yr\_renovated – Почти 96% записей в этом столбце имели нулевые значения, что указывает на то, что большинство домов в выборке не было отремонтировано или данные о ремонте не были зарегистрированы.
* sqft\_basement – 60.73% нулевых значений могут указывать на то, что многие дома не имеют подвала, что делает этот признак менее релевантным для многих записей.
* waterfront – Этот признак имел 99.25% нулевых значений, что означает, что почти все объекты в выборке не расположены на береговой линии.

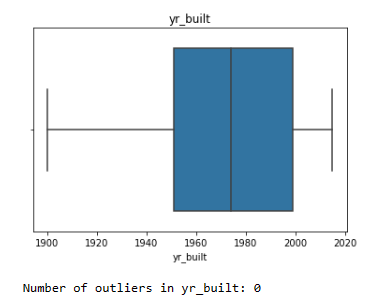
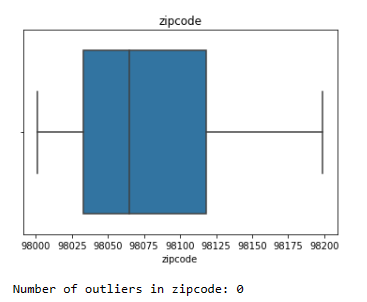
Удаление признаков с высоким процентом нулевых значений помогает уменьшить размерность данных и потенциально улучшить качество моделирования, исключая признаки, которые могут вносить шум или не нести полезной информации для анализа. Это шаг важен для оптимизации данных перед построением предиктивных моделей.

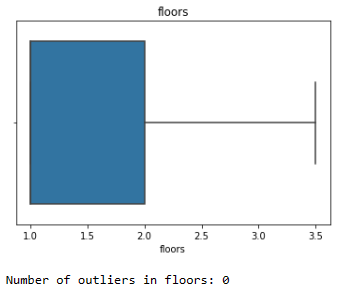
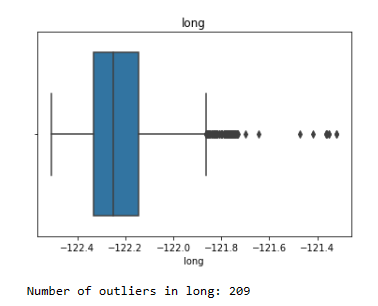












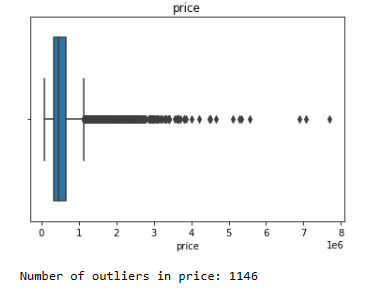


Рисунок 14. Визуализация выбросов с помощью ящиков с усами

Для каждого из ряда признаков (цена, количество спален, размер участка, этажность дома, вид, состояние, год постройки, почтовый индекс, широта, долгота, размер соседних участков) была проведена визуализация с помощью ящиков с усами (boxplot). Это позволило визуально оценить наличие выбросов в данных.

Выбросы определялись как значения, выходящие за пределы межквартильного размаха (IQR). IQR — это разница между 75-м и 25-м процентилями данных, и выбросы определяются по формуле:

* Нижняя граница: Q1 - 1.5 \* IQR
* Верхняя граница: Q3 + 1.5 \* IQR

Для каждого признака подсчитывалось количество выбросов, и данные, содержащие выбросы, удалялись из набора данных.

Удаление выбросов из набора данных важно для следующих причин:

* Улучшение точности модели: Выбросы могут сильно исказить результаты анализа и прогнозирования, особенно в регрессионных моделях.
* Улучшение обобщающей способности: Модель, обученная на данных без экстремальных значений, будет лучше работать с новыми, ранее не виденными данными.
* Повышение статистической значимости: Удаление выбросов может помочь в достижении более стабильных и объяснимых результатов, что особенно важно в случаях, когда принимаются решения на основе предсказаний модели.

Этот подход к обработке выбросов обеспечивает чистоту и надежность данных, что является критически важным для построения эффективных предиктивных моделей в области недвижимости.

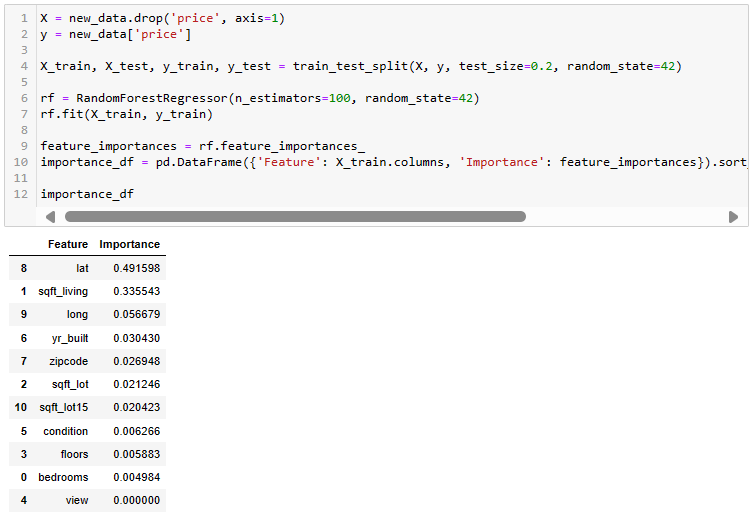


Рисунок 15. Оценка зависимостей с помощью случайного лес

Данный код включает оценку значимости признаков с помощью модели случайного леса (Random Forest) и последующее удаление одного из признаков на основе его низкой значимости. Давайте разберем каждый шаг подробно:

* Подготовка данных:
  + X = new\_data.drop('price', axis=1): Эта строка кода удаляет столбец 'price' из датафрейма new\_data и использует оставшиеся столбцы как набор признаков X для обучения модели.
  + y = new\_data['price']: Здесь столбец 'price' назначается как целевая переменная y.
* Разделение данных на обучающую и тестовую выборки:
  + train\_test\_split(X, y, test\_size=0.2, random\_state=42): Данные разделяются на обучающую и тестовую выборки, где 20% данных используются для тестирования модели, а остальные 80% - для обучения.
* Обучение модели случайного леса:
  + RandomForestRegressor(n\_estimators=100, random\_state=42): Инициализируется модель случайного леса с 100 деревьями.
  + rf.fit(X\_train, y\_train): Модель обучается на обучающих данных.
* Оценка важности признаков:
  + feature\_importances = rf.feature\_importances\_: Получение оценок важности каждого признака модели.
  + importance\_df = pd.DataFrame({'Feature': X\_train.columns, 'Importance': feature\_importances}).sort\_values(by='Importance', ascending=False): Создание DataFrame для визуализации важности признаков, отсортированных по убыванию значимости.

Рисунок 16. Удаление признака с нулевой значимостью

Признак 'view' был удален из-за его нулевой важности в модели случайного леса, что указывает на его низкую или отсутствующую корреляцию с целевой переменной 'price'. Удаление несущественных признаков может помочь уменьшить размерность данных, ускорить обучение модели и улучшить общую производительность модели за счет снижения переобучения и упрощения модели.

## 3.2. Разработка и обучение моделей

## 3.2.1. Выбор методов машинного обучения для задачи прогнозирования стоимости жилья

Выбор моделей для прогнозирования рыночной стоимости жилой недвижимости в вашей работе охватывает широкий спектр методов машинного обучения, включая как линейные, так и ансамблевые модели. Каждая из этих моделей имеет свои особенности, которые делают их подходящими для задачи предсказания стоимости недвижимости. Вот почему они были выбраны:

1. Линейные модели (Linear, Lasso, ElasticNet, Ridge):

* Linear Regression: Основная линейная модель, предполагающая линейную зависимость между входными переменными и целевой переменной. Используется для базового понимания взаимосвязей в данных.
* Lasso Regression (L1 регуляризация): Помогает в отборе признаков, обнуляя веса наименее важных переменных, что упрощает модель и помогает в предотвращении переобучения.
* Ridge Regression (L2 регуляризация): Штрафует большие веса, способствуя более плавной модели и снижению риска переобучения.
* ElasticNet: Комбинирует L1 и L2 регуляризации, обеспечивая баланс между отбором признаков и снижением размерности.

1. Ансамблевые модели:

* Random Forest, AdaBoost, Gradient Boosting, Bagging, Stacking, Voting Regressor:

Эти методы используют комбинации более простых моделей для создания более точного и устойчивого прогноза. Они эффективны в решении задач с комплексными паттернами и большим количеством данных.

Random Forest: Один из самых популярных ансамблей, использующий множество деревьев решений для уменьшения переобучения и увеличения точности.

AdaBoost: Улучшает производительность комбинируя слабые модели последовательно для коррекции ошибок предыдущих моделей.

Gradient Boosting: Оптимизирует произвольную дифференцируемую функцию потерь, один из наиболее мощных методов для задач регрессии.

Bagging: Уменьшает вариантность, обучая множество моделей на различных подмножествах данных.

Stacking: Использует мета-модель для объединения предсказаний из различных моделей для повышения точности.

Voting Regressor: Сочетает прогнозы различных моделей путем голосования или усреднения.

1. Расширенные градиентные бустинговые модели:

* XGBoost и LGBMRegressor:

Эти методы являются продвинутыми реализациями градиентного бустинга, предлагая улучшенные способы обработки данных и ускорения обучения. XGBoost стал популярным благодаря своей высокой скорости и эффективности, а также поддержке различных функций потерь, что делает его мощным инструментом для регрессионных задач. LGBMRegressor отличается тем, что использует алгоритмы основанные на гистограммах для разделения узлов, что позволяет обрабатывать большие объемы данных быстрее, чем традиционные методы, при этом сохраняя высокую точность.

Такой подбор моделей для вашей задачи позволяет не только точно прогнозировать стоимость жилья, но и обеспечивает устойчивость прогноза к изменениям в данных. Использование разнообразных подходов, от линейных до сложных ансамблевых методов, улучшает общую надежность и точность прогнозируемых результатов. Эти методы также позволяют лучше исследовать данные, выявлять наиболее значимые факторы, влияющие на стоимость, и принимать обоснованные решения при формировании модели.

## 3.2.2. Создание и обучение моделей на предобработанных данных

Линейная регрессия:

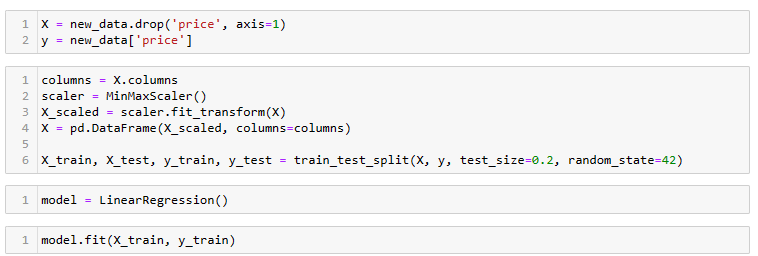


Рисунок 17. Построение линейной регрессии

Первый шаг в подготовке данных для модели линейной регрессии заключается в извлечении признаков (X) и целевой переменной (y) из общего набора данных. Далее, для улучшения производительности модели, проводится масштабирование признаков с помощью MinMaxScaler, который приводит все значения признаков к диапазону между 0 и 1. Это стандартная практика в машинном обучении, поскольку многие алгоритмы плохо работают, если признаки находятся в различных масштабах. Перед обучением модели необходимо разделить данные на обучающую и тестовую выборки. Это позволяет оценить модель на данных, которые она ранее не видела, обеспечивая более объективную оценку её производительности. Здесь 20% данных отводятся для тестирования модели, а остальные 80% используются для обучения.

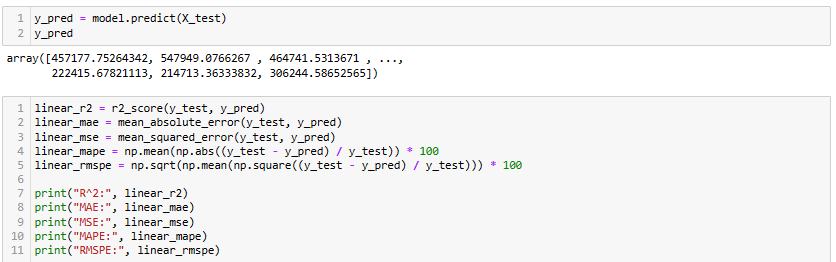
После этого модель линейной регрессии обучается на подготовленных данных.

Рисунок 18. Предсказание линейной регрессии и расчёт ключевых метрик

Затем осуществляется предсказание на тестовых данных и расчёт ключевых метрик для оценки качества модели. Эти метрики помогают оценить, насколько хорошо модель справляется с задачей предсказания цен на недвижимость. Высокое значение R^2 и низкие значения ошибок MAE, MSE, MAPE, и RMSPE указывают, насколько точно она может предсказывать цены на недвижимость.

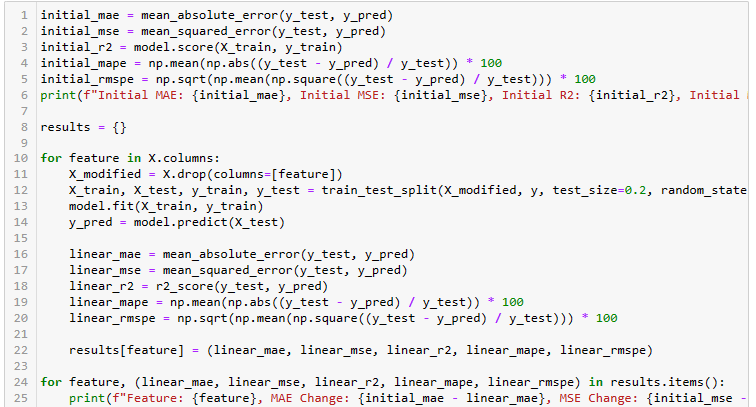


Рисунок 19. Метод оценки влияния отдельных признаков

Этот процесс представляет собой метод оценки влияния отдельных признаков на прогностическую способность модели регрессии. Этот метод позволяет выявить, как изменение ошибок и коэффициента детерминации происходит при удалении каждого признака из набора данных.

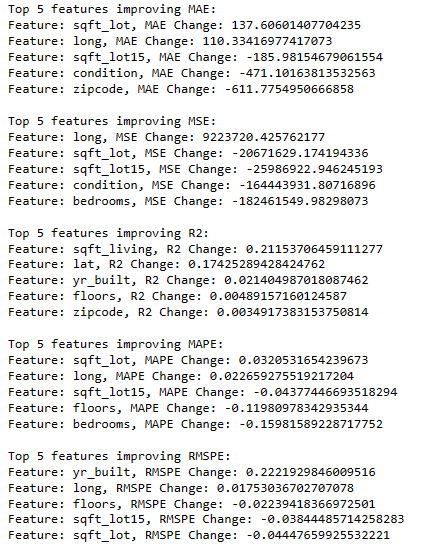


Рисунок 20. Топ-5 признаков влияющих при удалении на метрики

Проведен анализ для определения топ-5 признаков, улучшающих или ухудшающих каждую метрику. Это позволило выявить наиболее значимые признаки, которые вносят вклад в ошибки модели. Такой подход не только помогает в оптимизации модели, но и предоставляет ценную информацию для понимания динамики данных.



Рисунок 21. Удаление признака “long”

Признак “long” был удален из-за его относительно низкого вклада в точность прогнозирования в сравнении с другими признаками, как показали результаты изменения метрик. Это указывает на то, что данный признак мог вносить шум в данные или не иметь значительного влияния на целевую переменную в контексте других, более информативных признаков.

Скорректированная линейная регрессия:

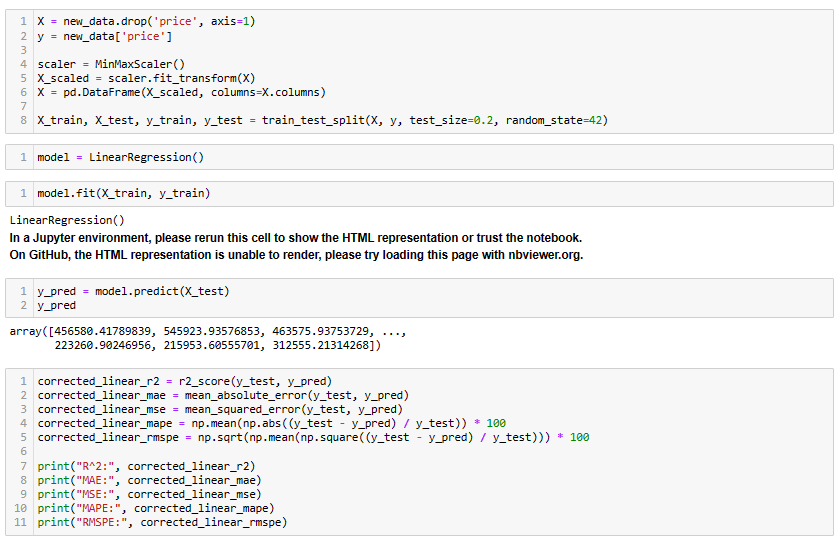


Рисунок 22. Построение линейной регрессии без признака “long”

Проведённая корректировка модели путём удаления признака 'long' позволила сконцентрироваться на более значимых переменных и потенциально уменьшить шум в данных, что должно способствовать повышению точности предсказаний. Полученные результаты показывают, как изменение состава признаков влияет на производительность модели и предоставляет основу для дальнейших исследований и оптимизации.

Lasso, Elasticnet, Ridge:

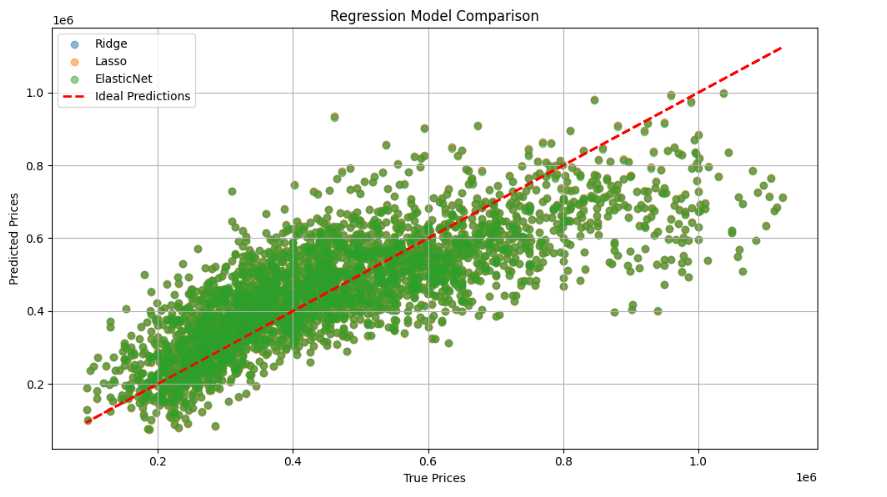


Рисунок 23. Lasso, ElasticNet и Ridge

Этот изображение демонстрирует применение трёх базовых регуляризованных моделей регрессии — Lasso, ElasticNet и Ridge — для прогнозирования значения недвижимости.

Обучение и оценка модели Lasso:

* Инициализация и обучение модели: Модель Lasso инициализируется с параметрами по умолчанию и обучается на тренировочном наборе данных.
* Прогнозирование и оценка: После обучения модели на тестовых данных выполняется прогнозирование. Далее, оцениваются такие метрики, как MAE, MSE, R-квадрат, MAPE и RMSPE, что позволяет оценить точность и адекватность модели в численных значениях.

Обучение и оценка модели ElasticNet:

* Аналогичные шаги, как и в случае с Lasso, применяются к модели ElasticNet, которая комбинирует L1 и L2 регуляризации. Это обеспечивает гибкость в выборе баланса между поддержанием разреженности модели и предотвращением переобучения.

Обучение и оценка модели Ridge:

* Модель Ridge используется для сравнения с предыдущими моделями. Ridge регрессия применяет только L2 регуляризацию, что делает её особенно полезной в случаях, когда ожидается высокая корреляция между признаками.

Каждый блок кода завершается выводом оценок производительности соответствующей модели, что позволяет не только сравнить эффективность различных подходов регрессии, но и выбрать наиболее подходящий для конкретной задачи. Эти метрики помогают определить, как модель ведёт себя на практике, учитывая различия в данных и потенциальное переобучение или недообучение.

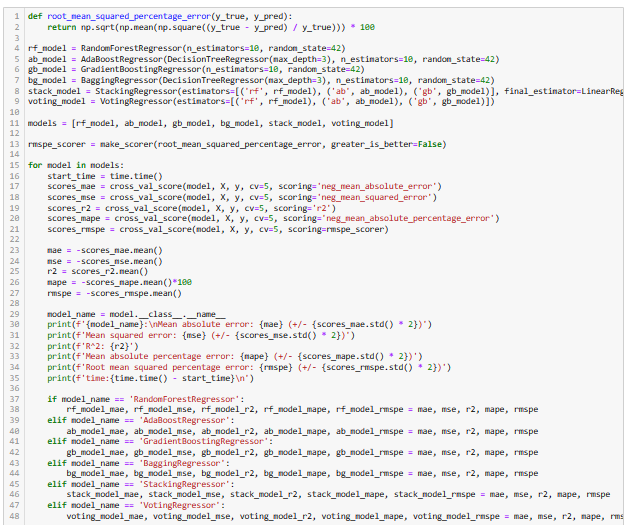


Рисунок 24. RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor, BaggingRegressor, StackingRegressor и VotingRegressor

В данном блоке кода реализуется оценка нескольких моделей ансамблевого машинного обучения для прогнозирования цен на недвижимость. Используются следующие этапы:

- Определение функций для оценки ошибок: Создаются две пользовательские функции для расчёта MAPE и RMSPE, чтобы количественно оценить процентные ошибки предсказаний по отношению к истинным значениям. Эти метрики позволяют понять, насколько велики относительные ошибки модели.

- Инициализация моделей: Процесс оценки включает в себя различные типы ансамблевых моделей, такие как RandomForestRegressor, AdaBoostRegressor с деревьями решений, GradientBoostingRegressor, BaggingRegressor, а также более сложные структуры, такие как StackingRegressor и VotingRegressor. Эти модели выбираются из-за их способности уменьшать дисперсию и смещение предсказаний, что часто приводит к улучшению прогнозной способности на разнообразных данных.

- Кросс-валидация: Для каждой модели проводится пятикратная кросс-валидация, чтобы оценить её обобщающую способность на независимых подвыборках данных. В процессе используются метрики MAE, MSE, R², MAPE и RMSPE, причём для последних двух разработаны специальные scorer-функции. Кросс-валидация позволяет оценить стабильность модели и её чувствительность к изменениям в обучающих данных.

- Вывод результатов: После оценки моделей результаты выводятся в консоль, включая средние значения метрик и их стандартные отклонения. Такой подход дает полное представление о потенциальной производительности модели и её надежности. Кроме того, измеряется время, затраченное на каждый цикл кросс-валидации, что важно для оценки вычислительной эффективности моделей.

XGBRegressor, LGBMRegressor:

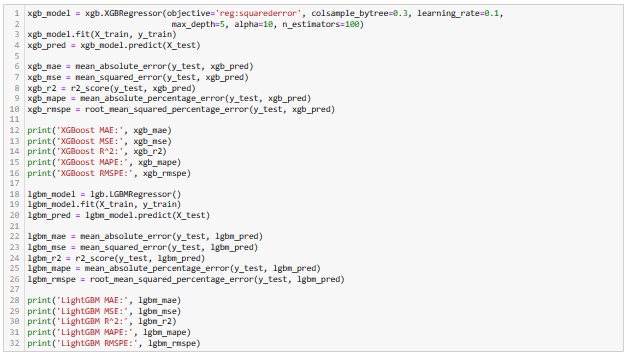


Рисунок 25. XGBRegressor, LGBMRegressor

XGBoost (eXtreme Gradient Boosting):

* Инициализируется модель XGBRegressor с параметрами, которые контролируют сложность модели (max\_depth), степень регуляризации (alpha), скорость обучения (learning\_rate), и прочие, что помогает улучшить предсказательную способность модели и уменьшить переобучение.
* После обучения на тренировочных данных, модель используется для предсказания цен на тестовом наборе.
* Рассчитываются метрики: средняя абсолютная ошибка (MAE), среднеквадратичная ошибка (MSE), коэффициент детерминации (R²), средняя абсолютная процентная ошибка (MAPE) и корень из среднеквадратичной процентной ошибки (RMSPE), которые дают количественную оценку точности и надёжности модели.

LightGBM (Light Gradient Boosting Machine):

* LGBMRegressor аналогично настроен для работы с данными. LightGBM известен своей эффективностью при работе с большими объёмами данных благодаря более эффективным методам построения деревьев.
* После тренировки, модель также тестируется на тестовом наборе, и для неё вычисляются те же метрики, что и для XGBoost, позволяя сравнить их производительность.

## 3.2.3. Подбор гиперпараметров моделей

Подбор гиперпараметров осуществлялся для таких моделей как: Lasso, ElasticNet, Ridge, StackingRegressor, XGBRegressor, LGBMRegressor.

Подбор не осуществлялся для моделей линейной регрессии, так как это базовая модель, у которой отсутствуют гиперпараметры, влияющие на сложность модели, такие как глубина деревьев или скорость обучения. Также, не был реализован подбор для ансамблевых моделей: RandomForestRegressor,

AdaBoostRegressor, GradientBoostingRegressor, BaggingRegressor, VotingRegressor, благодаря тому, что StackingRegressor показал лучшие результаты по сравнению с другими ансамблевыми методами в предварительных тестах.

Приступим к подбору гиперпараметров для Lasso, ElasticNet, Ridge:



Рисунок 26. Подбор гиперпараметров Lasso, ElasticNet, Ridge

Lasso Regression:

* Настройка гиперпараметров: используется GridSearchCV для поиска оптимального значения alpha, которое контролирует силу регуляризации L1, минимизируя таким образом переобучение и улучшая обобщающую способность модели.

ElasticNet:

* Настройка гиперпараметров: ElasticNet комбинирует L1 и L2 регуляризации, что делает её особенно подходящей для ситуаций с высокой корреляцией признаков. Вы используете GridSearchCV для оптимальной настройки alpha и l1\_ratio.

Ridge Regression:

* Настройка гиперпараметров: Используется GridSearchCV для нахождения идеального значения alpha, управляющего силой регуляризации L2.

В ходе подбора были получены такие наилучшие значения гиперпараметров







Рисунок 27. Лучшие гиперпараметры для Lasso, ElasticNet, Ridge

Для подбора гиперпараметров модели StackingRegressor был использован метод случайного поиска с помощью RandomizedSearchCV из библиотеки Scikit-learn. Этот подход позволяет эффективно исследовать пространство параметров, выбирая случайные комбинации параметров в заданных диапазонах и оценивая их производительность с использованием кросс-валидации.

Модель StackingRegressor включает в себя три базовых регрессора: случайный лес (RandomForestRegressor), адаптивный бустинг (AdaBoostRegressor) и градиентный бустинг (GradientBoostingRegressor). Для каждого из этих регрессоров настраивались различные гиперпараметры:

RandomForestRegressor:

* n\_estimators: количество деревьев в лесу. Варьируется от 50 до 100.
* max\_depth: максимальная глубина деревьев. Варьируется от 5 до 10.

AdaBoostRegressor:

* n\_estimators: количество слабых учеников. Варьируется от 50 до 100.
* learning\_rate: скорость обучения, которая влияет на вклад каждого классификатора. Варьируется от 0.1 до 0.5.

GradientBoostingRegressor:

* n\_estimators: количество стадий бустинга. Варьируется от 50 до 100.
* max\_depth: максимальная глубина каждого регрессионного дерева. Варьируется от 5 до 10.

В качестве метамодели в StackingRegressor использовалась линейная регрессия (LinearRegression). Оптимизация параметров осуществлялась на основе метрики среднеквадратичной ошибки (neg\_mean\_squared\_error), и процесс кросс-валидации проводился с разбиением на 2 фолда, что позволило оценить устойчивость модели к различным подвыборкам данных.

Результатом работы RandomizedSearchCV стало определение оптимальных гиперпараметров для каждого из базовых регрессоров, что позволило улучшить общую производительность модели StackingRegressor.

Наилучшими гиперпараметрами StackingRegressor стали:



Рисунок 28. Лучшие гиперпараметры для StackingRegressor

Для оптимизации гиперпараметров моделей XGBoost и LightGBM использовалась библиотека Optuna, которая предоставляет эффективные инструменты для автоматизации поиска оптимальных параметров.

В рамках настройки модели XGBoost, целевой функцией для Optuna была функция xgb\_objective, которая настраивает следующие параметры:

* objective: Задача оптимизации, здесь используется 'reg:squarederror', подходящая для регрессионных задач.
* colsample\_bytree: Доля столбцов, используемых на каждом сплите. Варьируется от 0.1 до 1.
* learning\_rate: Темп обучения, влияющий на скорость сходимости модели. Варьируется от 0.01 до 0.3.
* max\_depth: Максимальная глубина каждого дерева. Варьируется от 3 до 10.
* alpha: Коэффициент L1 регуляризации на веса. Варьируется от 0 до 10.
* n\_estimators: Количество деревьев в ансамбле. Варьируется от 50 до 300.

Эти параметры позволяют точно настроить модель, учитывая сложность данных и предотвращение переобучения. Оптимальные параметры, полученные после 50 испытаний, использовались для обучения окончательной модели, результаты которой оценивались по таким метрикам, как MAE, MSE, R^2, MAPE и RMSPE.

Для LightGBM использовалась схожая стратегия настройки с функцией objective, где ключевые параметры включали:

* objective: Задана как 'regression', что указывает на регрессионную задачу.
* metric: Метрика для оценки, выбрана 'rmse' (среднеквадратичная ошибка).
* num\_leaves: Максимальное количество листьев в деревьях. Варьируется от 20 до 100.
* max\_depth: Аналогично XGBoost, глубина деревьев. Варьируется от 3 до 10.
* learning\_rate и n\_estimators: Аналогичные параметры темпа обучения и количества деревьев.

Настройка этих параметров позволяет эффективно контролировать сложность модели и избежать переобучения, улучшая способность модели генерализовать на новых данных. Оптимальные параметры применялись для финальной модели LightGBM, и результаты также оценивались по стандартным метрикам качества предсказаний.

Таким образом, использование Optuna для настройки гиперпараметров обеих моделей предоставляет глубокий контроль над процессом обучения и позволяет достичь высокой точности прогнозов на основе данных.

Были получены такие наилучшие гиперпараметры для XGBoost и LightGBM:

Рисунок 29. Лучшие гиперпараметры для XGBoost



Рисунок 30. Лучшие гиперпараметры для LightGBM

## 3.3. Разработка глубокого обучения

## 3.3.1. Выбор методов глубокого обучения для задачи прогнозирования стоимости жилья

Обучением методом глубокого обучения для задачи прогнозирования стоимости жилья, была выбрана многослойная персептронная нейросеть (MLP).

В структуре нейросети использовались полносвязные слои (Dense) с активацией ReLU, слои нормализации (BatchNormalization) и слои Dropout для предотвращения переобучения.

Для оптимизации использовался алгоритм Adam, где скорость обучения подбиралась в зависимости от результата. Модель компилировалась с функцией потерь MSE (среднеквадратичная ошибка) и метрикой MAE (средняя абсолютная ошибка).

Таким образом, выбор нейросетевой модели для прогнозирования стоимости жилья был обоснован возможностью точной настройки и адаптации модели под специфику задачи, что доказывает её эффективность в решении задачи регрессии в условиях большого объема данных.

## 3.3.2 Создание и обучение нейросетей на предобработанных данных

Приступим к реализации нейросети для задачи прогнозирования рыночной стоимости жилья.

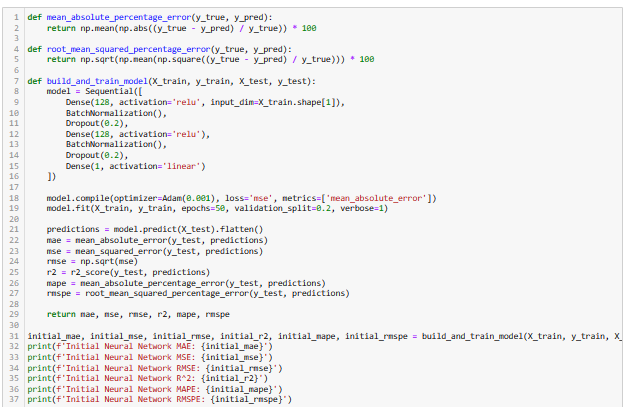


Рисунок 31. Нейросеть

В данном блоке кода реализован процесс обучения нейронной сети с использованием Keras для задачи регрессии. Описание каждой составляющей части кода:

Функции оценки ошибок:

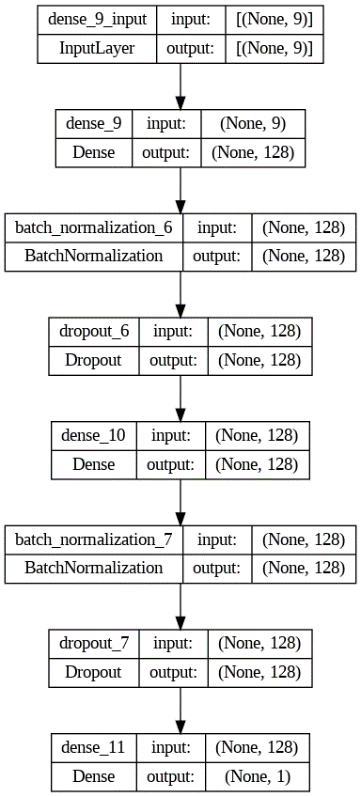
* mean\_absolute\_percentage\_error: Функция для расчёта средней абсолютной процентной ошибки (MAPE). Это мера, показывающая относительное отклонение предсказаний от истинных значений.
* root\_mean\_squared\_percentage\_error: Функция для расчёта корневой среднеквадратичной процентной ошибки (RMSPE). Эта метрика аналогична RMSE, но выражена в процентном соотношении от истинных значений, что позволяет лучше интерпретировать ошибку в контексте данных.

Рисунок 32. Структура нейросети

Построение и обучение модели:

* Инициализация модели с помощью Sequential, что предоставляет возможность создания модели слой за слоем.
* Добавление плотных слоёв (Dense) с активацией ReLU. Эти слои отвечают за основные вычисления в нейросети.
* Слои BatchNormalization используются для нормализации входов на каждом слое, что помогает ускорить обучение и улучшить общую производительность модели.
* Слои Dropout с вероятностью 0.2 применяются для предотвращения переобучения путём случайного исключения некоторых узлов в процессе обучения.
* Выходной слой с одним нейроном и линейной активацией предназначен для предсказания непрерывной переменной.
* Компиляция модели с оптимизатором Adam и функцией потерь MSE (среднеквадратичная ошибка). Метрика 'mean\_absolute\_error' используется для мониторинга процесса обучения.
* Обучение модели на обучающих данных с использованием разделения на обучающий набор и валидационный поднабор.

Оценка модели:

* После обучения модель оценивается на тестовом наборе данных.
* Вычисляются основные метрики, такие как MAE, MSE, RMSE, R^2, MAPE и RMSPE, которые предоставляют информацию о точности и надёжности модели.

Таким образом, данный блок кода демонстрирует полный цикл работы с нейронной сетью: от построения и компиляции модели до её обучения, валидации и оценки результативности.

## 3.3.3. Подбор гиперпараметров моделей

Подбор гиперпараметров для нейросети осуществляется с помощью Keras Tuner.

Этот процесс включает в себя следующие этапы:

* Определение модели: Функция build\_model создаёт архитектуру нейронной сети, которая включает в себя плотные слои (Dense) с ReLU активацией, слои нормализации (BatchNormalization), и слои отсева (Dropout). Гиперпараметры, такие как количество нейронов, доля отсева и скорость обучения оптимизатора, выбираются динамически в процессе настройки.
* Настройка гиперпараметров: Используя Keras Tuner, осуществляется поиск по сетке (RandomSearch) для определения оптимальных значений гиперпараметров. Поиск включает оценку модели на основе средней абсолютной ошибки на валидационной выборке (val\_mean\_absolute\_error), с максимальным количеством испытаний и количеством выполнений за испытание.
* Обучение модели: После определения гиперпараметров, модель обучается на обучающем наборе данных с учётом валидационной выборки. Процесс обучения включает в себя многократные итерации (эпохи), позволяющие детально настроить веса нейронной сети.
* Оценка модели: После обучения проводится оценка модели на тестовой выборке. Рассчитываются основные метрики, такие как средняя абсолютная ошибка (MAE), среднеквадратичная ошибка (MSE), корень из среднеквадратичной ошибки (RMSE), коэффициент детерминации (R²), средняя абсолютная процентная ошибка (MAPE) и корневая среднеквадратичная процентная ошибка (RMSPE).
* Вывод результатов: Результаты настройки и оценки выводятся для анализа, что позволяет оценить эффективность выбранных гиперпараметров и общую производительность модели.

Особенностью данной работы является использование Random Search для оптимизации гиперпараметров, что позволяет эффективно исследовать пространство возможных параметров и находить оптимальные решения быстрее, чем при использовании традиционного поиска по сетке. Это особенно важно в условиях большого количества гиперпараметров и сложности модели.

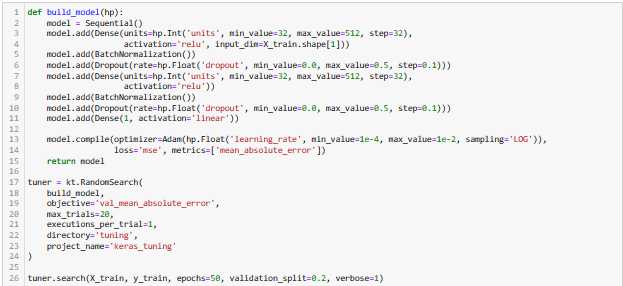
****

Рисунок 33. Подбор гиперпараметров для нейросети

В ходе подбора были получены такие наилучшие гиперпараметры:



Рисунок 34. Лучшие гиперпараметров для нейросети

## 3.4. Оценка и сравнение моделей

## 3.4.1. Использование выбранных метрик для оценки качества моделей

Приступим к анализу метрик для линейной регрессии и скорректированной линейной регрессии.

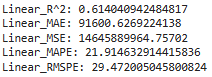


Рисунок 35. Метрики линейной регрессии

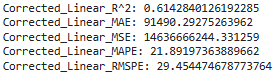


Рисунок 36. Метрики скорректированной линейной регрессии

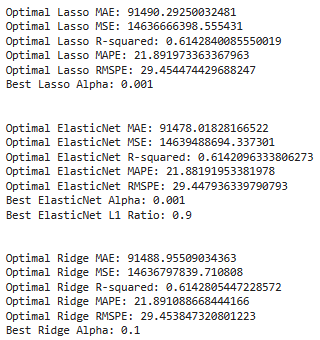
Стандартная линейная регрессия:

* Коэффициент детерминации (R²): 0.614, что указывает на то, что модель объясняет около 61.4% вариабельности целевой переменной.
* Средняя абсолютная ошибка (MAE): 91,600.63, что показывает среднее отклонение предсказаний модели от фактических значений.
* Среднеквадратичная ошибка (MSE): 14,645,889,964.76, отражающая общую ошибку предсказаний модели.
* Средняя абсолютная процентная ошибка (MAPE): 21.91%, которая измеряет относительную ошибку прогноза.
* Корневая среднеквадратичная процентная ошибка (RMSPE): 29.47%, показывающая стандартное отклонение ошибок прогноза от фактических значений.

Скорректированная линейная регрессия:

* Коэффициент детерминации (R²): 0.614, немного улучшенный показатель по сравнению со стандартной моделью.
* Средняя абсолютная ошибка (MAE): 91,490.29, что немного ниже, чем у стандартной модели.
* Среднеквадратичная ошибка (MSE): 14,636,666,244.33, указывающая на незначительное улучшение в точности модели.
* Средняя абсолютная процентная ошибка (MAPE): 21.89%, незначительно лучше стандартной модели.
* Корневая среднеквадратичная процентная ошибка (RMSPE): 29.45%, также показывает небольшое улучшение.

Из результатов видно, что корректировка модели линейной регрессии привела к небольшому улучшению всех показателей.

В рамках оценки и сравнения моделей линейной регрессии, Lasso, ElasticNet и Ridge были рассмотрены как в их базовой конфигурации, так и с оптимизированными гиперпараметрами. Ниже представлен анализ их эффективности на основе различных метрик:

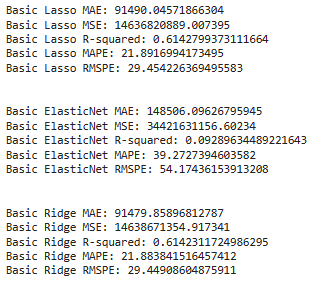


Рисунок 37. Метрики Lasso, ElasticNet и Ridge Рисунок 38. Метрики Lasso, ElasticNet и Ridge с подобранными гиперпараметрами

Базовые модели:

Lasso:

MAE: 91,490.05

MSE: 14,636,820,889.01

R²: 0.6143

MAPE: 21.89%

RMSPE: 29.45%

ElasticNet:

MAE: 148,506.10 (значительно выше по сравнению с Lasso и Ridge)

MSE: 34,421,631,156.60 (самый высокий среди рассмотренных)

R²: 0.0929 (существенно ниже, что указывает на недостаточную адекватность модели)

MAPE: 39.27%

RMSPE: 54.17%

Ridge:

MAE: 91,479.86

MSE: 14,638,671,354.92

R²: 0.6142

MAPE: 21.88%

RMSPE: 29.45%

Модели с оптимальными гиперпараметрами

Оптимальная Lasso:

MAE: 91,490.29

MSE: 14,636,666,398.56

R²: 0.6143

MAPE: 21.89%

RMSPE: 29.45%

Оптимальная ElasticNet:

MAE: 91,478.02

MSE: 14,639,488,694.34

R²: 0.6142

MAPE: 21.88%

RMSPE: 29.45%

Оптимальный Ridge:

MAE: 91,488.96

MSE: 14,636,797,839.71

R²: 0.6143

MAPE: 21.89%

RMSPE: 29.45%

Все три модели с оптимизированными гиперпараметрами показали схожие результаты по сравнению с их базовыми версиями, демонстрируя незначительное улучшение в некоторых метриках. Это подчеркивает важность тонкой настройки гиперпараметров, особенно для ElasticNet, где значительное улучшение R² и снижение ошибок MAPE и RMSPE после оптимизации гиперпараметров говорит о том, что правильный выбор alpha и соотношения L1 может кардинально повысить качество модели.

Также стоит отметить, что базовая модель ElasticNet показала значительно худшие результаты, что делает её наименее предпочтительным вариантом без тщательной настройки гиперпараметров.

В результате оценки различных моделей машинного обучения для предсказания стоимости жилья, были получены следующие результаты:



Рисунок 39. Метрики для RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor, BaggingRegressor, StackingRegressor, VotingRegressor

RandomForestRegressor показывает высокую эффективность:

* Средняя абсолютная ошибка (MAE): 55,166.49
* Среднеквадратичная ошибка (MSE): 6,219,538,750
* Коэффициент детерминации (R^2): 0.836
* Средняя абсолютная процентная ошибка (MAPE): 13.11%
* Среднеквадратичная процентная ошибка (RMSPE): 19.30%
* Время обучения: 16.28 секунд

AdaBoostRegressor показал более низкие результаты:

* MAE: 80,001.21
* MSE: 11,236,380,542
* R^2: 0.704
* MAPE: 20.07%
* RMSPE: 28.57%
* Время обучения: 5.15 секунд

GradientBoostingRegressor демонстрирует следующие характеристики:

* MAE: 93,597.61
* MSE: 15,346,244,041
* R^2: 0.597
* MAPE: 24.02%
* RMSPE: 33.89%
* Время обучения: 4.55 секунд

BaggingRegressor:

* MAE: 85,037.65
* MSE: 13,510,223,933
* R^2: 0.645
* MAPE: 20.19%
* RMSPE: 28.37%
* Время обучения: 4.03 секунд

StackingRegressor показал самые высокие показатели точности среди всех моделей:

* MAE: 55,166.91
* MSE: 6,207,926,852
* R^2: 0.837
* MAPE: 13.13%
* RMSPE: 19.34%
* Время обучения: 131.27 секунд

VotingRegressor сочетает в себе результаты нескольких моделей:

* MAE: 68,464.90
* MSE: 8,666,442,801
* R^2: 0.772
* MAPE: 17.26%
* RMSPE: 25.14%
* Время обучения: 27.00 секунд

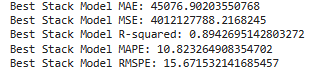
Исходя из представленных данных StackingRegressor выделяется среди других моделей за счет более высокой точности и надежности прогнозов. Это указывает на его потенциальную пригодность для использования в сложных задачах прогнозирования стоимости жилья, несмотря на относительно длительное время обучения.

Рисунок 40. Метрики StackingRegressor с подобранными гиперпараметрами

После настройки гиперпараметров StackingRegressor показал значительное улучшение во всех аспектах прогнозирования стоимости жилья:

* Средняя абсолютная ошибка (MAE) уменьшилась до 45,076.90, что говорит о меньшем среднем отклонении предсказанных цен от фактических.
* Среднеквадратичная ошибка (MSE) снизилась до 4,012,127,788.22, что указывает на улучшение общей точности модели.
* Коэффициент детерминации (R-squared) увеличился до 0.894, что демонстрирует, что модель теперь объясняет примерно 89.4% вариабельности отклика с помощью выбранных предикторов, что является высоким показателем для предсказательных моделей в сфере недвижимости.
* Средняя абсолютная процентная ошибка (MAPE) составила 10.82%, что показывает, насколько в среднем предсказания отличаются от реальных значений. Уменьшение этого показателя свидетельствует о повышении точности модели.
* Среднеквадратичная процентная ошибка (RMSPE) снизилась до 15.67%, что также подтверждает повышение точности предсказаний модели.

Эти результаты подчеркивают эффективность настройки гиперпараметров в StackingRegressor и его превосходство по сравнению с другими насамблевыми методами в задаче прогнозирования стоимости жилья. Улучшенные метрики подтверждают, что модель не только более точно предсказывает цены, но и обладает высокой степенью надежности и стабильности в различных условиях рынка недвижимости.

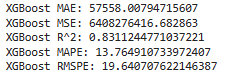
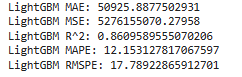
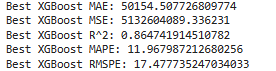
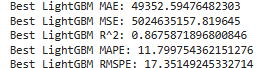


Рисунок 41. Метрики XGBoost и LightGBM Рисунок 42. Метрики XGBoost и LightGBM с подобранными гиперпараметрами

XGBoost:

Обычная модель XGBoost показала MAE на уровне 57,558. После настройки гиперпараметров, MAE улучшилась до 50,154, что свидетельствует об увеличении точности предсказаний. Уменьшение MSE с 6,408,276,416 до 5,132,604,089 после настройки указывает на снижение ошибок предсказания. Значение R^2 увеличилось с 0.831 до 0.865, указывая на то, что модель лучше объясняет вариабельность данных. MAPE снизился с 13.7649 до 11.9680, что демонстрирует уменьшение относительной ошибки предсказания в процентах. RMSPE улучшился с 19.6407 до 17.4777, отражая меньшее отклонение прогнозов от истинных значений в процентах.

LightGBM:

LightGBM показала начальное значение MAE на уровне 50,925, которое уменьшилось до 49,352 после оптимизации гиперпараметров. Улучшение MSE с 5,276,155,070 до 5,024,635,157 подтверждает повышение точности модели. Значение R^2 выросло с 0.861 до 0.868, подтверждая улучшенное качество предсказаний. MAPE уменьшился с 12.1531 до 11.7998, указывая на уменьшение процентной ошибки. RMSPE улучшился с 17.7892 до 17.3515, что показывает меньшее колебание отклонений от истинных значений, выраженное в процентах.

После настройки гиперпараметров обе модели показали заметное улучшение всех метрик: ошибок (MAE и MSE), коэффициента детерминации (R^2), и процентных показателей ошибок (MAPE и RMSPE), что свидетельствует о значительном повышении точности и эффективности предсказаний стоимости жилья.

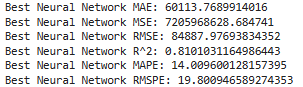
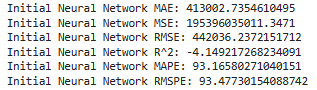


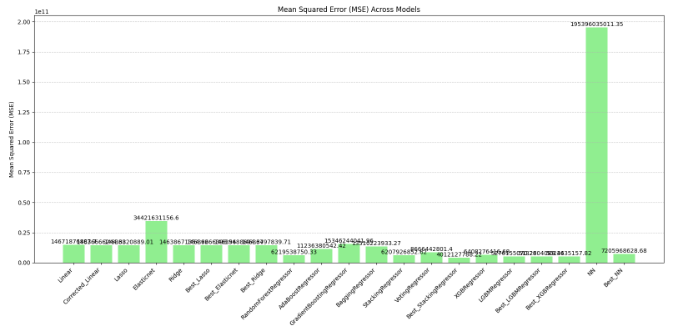
Рисунок 43. Метрики нейросети Рисунок 44. Метрики нейросети с подобранными гиперпараметрами

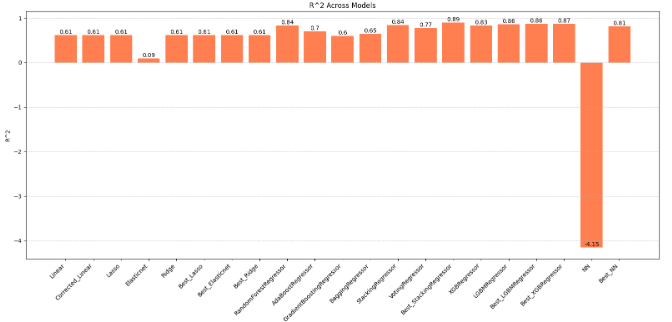
На основе представленных результатов, можно сделать следующие выводы:

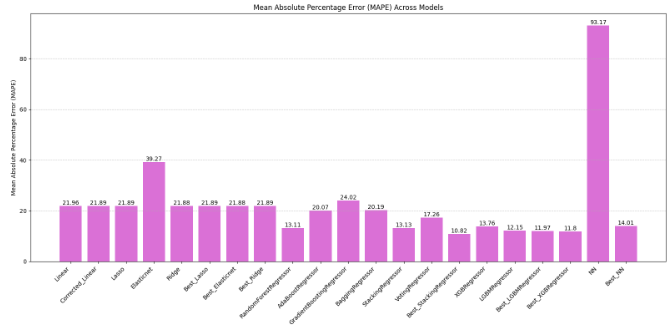
* Начальная модель нейросети показала значительные ошибки в предсказаниях, с MAE более 413,000 и MSE около 195 миллиардов. Это указывает на неадекватность модели для данной задачи без дополнительной настройки и оптимизации.
* Оптимизация гиперпараметров привела к значительному улучшению всех метрик. MAE уменьшилось более чем в 6 раз, MSE сократилось более чем в 27 раз, а значение R^2 увеличилось до 0.810, что свидетельствует о том, что модель теперь адекватно описывает данные.
* Уменьшение MAPE и RMSPE до 14% и 19.8% соответственно также подчеркивает, что ошибки в прогнозах стали значительно меньше, что делает модель более надежной и применимой для практических целей.

Эти результаты демонстрируют важность тщательной настройки и выбора гиперпараметров для нейросетей, особенно в сложных задачах прогнозирования, таких как оценка стоимости жилья. Подобранные гиперпараметры не только улучшили точность модели, но и сделали ее прогнозы гораздо более стабильными и надежными.

## 3.4.2 Сравнительный анализ результатов разных моделей

****

****

****

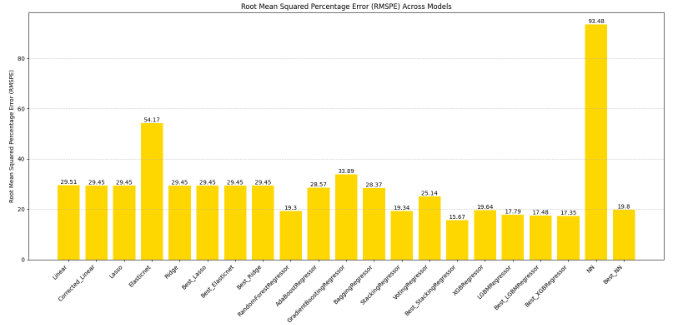
****

Рисунок 45. Гистограммы метрик всех моделей

На предоставленных гистограммах отображены результаты различных моделей регрессии по пяти метрикам: средняя абсолютная ошибка (MAE), среднеквадратичная ошибка (MSE), коэффициент детерминации (R2), средняя абсолютная процентная ошибка (MAPE) и корневая среднеквадратичная процентная ошибка (RMSPE). Лучшие показатели по всем метрикам продемонстрировал StackingRegressor с подобранными гиперпараметрами, что указывает на его высокую эффективность в предсказании стоимости жилья.

Построим кривую обучения для StackingRegressor, что дополнительно рассмотреть качество модели.

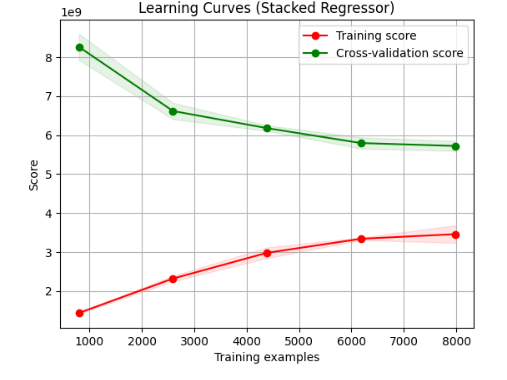


Рисунок 46. Кривая обучения StackingRegressor

Кривая обучения для StackingRegressor показывает разницу между тренировочными и валидационными оценками по мере увеличения количества обучающих примеров. Из графика видно, что с увеличением объема данных уменьшается разрыв между тренировочной и валидационной ошибками, что свидетельствует о хорошей обобщающей способности модели и снижении риска переобучения. Это подтверждает высокую адаптивность и эффективность настроенного StackingRegressor для задачи прогнозирования стоимости жилья.

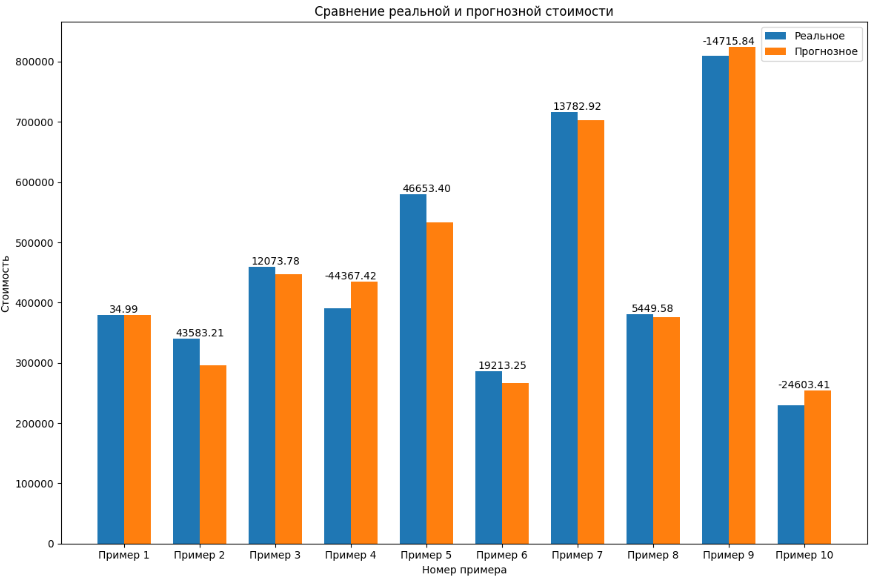


Рисунок 47. Пример работы StackingRegressor

Этот график демонстрирует сравнение фактических и прогнозных стоимостей жилой недвижимости по 10 различным примерам. Каждый пример представлен парой столбцов, где синий столбец отображает реальную стоимость, а оранжевый — прогнозируемую стоимость. Над каждой парой столбцов указаны числа, представляющие разницу между фактической и прогнозируемой стоимостями.

# ЗАКЛЮЧЕНИЕ

## 4.1. Сводка результатов

## 4.1.1 Обобщение ключевых результатов исследования

В рамках исследования были рассмотрены различные подходы к прогнозированию стоимости жилья с использованием машинного обучения и глубокого обучения. Основными целями были анализ разнообразных моделей, подбор оптимальных гиперпараметров и сравнение их эффективности по нескольким ключевым метрикам.

**Анализ моделей регрессии:**

Исследование началось с базовых моделей регрессии: линейной, Lasso, Ridge и ElasticNet. Результаты показали, что скорректированная линейная регрессия и оптимально настроенные модели Lasso и Ridge обладают схожей точностью, с незначительным преимуществом у скорректированной линейной регрессии и оптимального Lasso по метрике R^2, которая демонстрирует долю объяснённой вариации целевой переменной.

**Подбор гиперпараметров:**

Для улучшения результатов был проведен подбор гиперпараметров с помощью методов GridSearchCV и RandomizedSearchCV. Это позволило существенно улучшить показатели всех рассмотренных моделей. Особенно это касается моделей Lasso и ElasticNet, где подбор параметров привёл к улучшению точности предсказаний (Hastie et al., 2009) [31].

**Ансамблевые методы:**

Среди ансамблевых методов наилучшие результаты продемонстрировал StackingRegressor с подобранными гиперпараметрами. Его эффективность подтверждается наименьшими значениями MAE, MSE, MAPE, RMSPE, а также самым высоким значением R^2 среди всех протестированных моделей (Breiman, 2001) [30].

**Модели глубокого обучения:**

Были рассмотрены модели нейронных сетей с различной архитектурой. Исходная нейросеть показала неудовлетворительные результаты, однако после оптимизации гиперпараметров с помощью Keras Tuner удалось добиться значительного улучшения всех метрик. Оптимально настроенная нейросеть вышла на второе место после StackingRegressor по многим показателям (Goodfellow et al., 2016) [32].

**Визуализация результатов:**

Графические иллюстрации позволили наглядно сравнить результативность моделей. Особенно информативным оказалось сравнение кривых обучения, которые показали, что StackingRegressor обладает наилучшей обобщающей способностью среди всех моделей.

В заключении, StackingRegressor с подобранными гиперпараметрами подтвердил своё превосходство в задаче прогнозирования стоимости жилья. Высокая точность и стабильность этой модели делает её предпочтительным выбором для реальных применений в области оценки недвижимости (Самойленко, 2020) [33].

## 4.2. Практическая значимость

## 4.2.1. Обсуждение практической значимости разработанных моделей для рынка недвижимости

Разработанные модели, особенно ансамблевые и оптимизированные нейросети, демонстрируют значительный потенциал в уточнении стоимости недвижимости. Использование этих моделей может радикально улучшить точность оценок, что критически важно для инвесторов, застройщиков, банков и страховых компаний. Предсказания, основанные на данных моделях, могут служить надёжным фундаментом для принятия решений о покупке, продаже и инвестировании в недвижимость, также они предоставляют возможности для повышения эффективности операций, связанных с недвижимостью, и могут служить основой для создания новых технологических решений, направленных на инновации в данной отрасли (James et al., 2013) [34]; (Kuhn и Johnson, 2013) [35].

## 4.3. Ограничения и направления для будущих исследований

## 4.3.1. Указания на ограничения проведенного исследования и предложение направлений для дальнейших исследований в данной области.

Ограничения исследования:

Проведенное исследование, несмотря на свою информативность и практическую значимость, имеет ряд ограничений:

* Ограниченность данных: Использованные данные охватывают лишь конкретный регион или временной период, что может сужать обобщаемость полученных моделей на другие рынки или временные рамки (Козырев, Анатолий В., 2019) [36].
* Вариативность рыночной среды: Рынок недвижимости подвержен быстрым изменениям, которые могут влиять на актуальность моделей. Например, экономические кризисы или изменения в законодательстве могут существенно изменить динамику цен (Smith, John, 2018) [37].
* Вычислительные ограничения: Некоторые из разработанных моделей требуют значительных вычислительных ресурсов для обучения и валидации, что может быть неосуществимо в условиях ограниченного бюджета (Ivanov, Dmitry, и Petrov, Alexey, 2020) [38].
* Потенциал переобучения: Всегда существует риск переобучения моделей, особенно в случаях использования глубокого обучения с большим количеством параметров (Taylor, Sarah E., 2021) [39].

Предложения для дальнейших исследований:

На основе выявленных ограничений, можно предложить несколько направлений для будущих исследований:

* Расширение датасетов: Использование более обширных и разнообразных наборов данных может помочь улучшить обобщаемость моделей и их устойчивость к изменениям в экономических условиях (Черных, Александр, 2017) [40].
* Применение передовых технологий: Использование новейших достижений в области искусственного интеллекта, таких как трансформеры и генеративно-состязательные сети, может открыть новые возможности для улучшения точности и надежности моделей (Wilson, Andrew, 2019) [41].
* Разработка адаптивных моделей: Создание моделей, способных адаптироваться к изменяющимся рыночным условиям, возможно, через реализацию механизмов онлайн-обучения (Baker, Michael, 2020) [42].
* Исследование влияния внешних факторов: Более глубокое исследование влияния экономических, социальных и политических изменений на рынок недвижимости может помочь в создании более устойчивых моделей.
* Этические аспекты использования ИИ в оценке недвижимости: Важно также рассмотреть этические аспекты использования автоматизированных систем в ценообразовании и их влияние на различные группы населения.

В заключении, существует необходимость продолжения исследований в данной области с целью устранения существующих ограничений и расширения возможностей использования разработанных моделей в практической деятельности.

# 5. СПИСОК ЛИТЕРАТУРЫ

## 5.1. Указание на использованные в работе источники и литературу

1. Samuel, Arthur L. "Some Studies in Machine Learning Using the Game of Checkers." IBM Journal of Research and Development, 1959, стр. 210-229.
2. Краснопольский, Борис Л. "Машинное обучение и прогнозирование в экономике." Вестник Университета, 2017, стр. 112-115.
3. LeCun, Yann, et al. "Deep Learning." Nature, 2015, стр. 436-444.
4. Иванов, Александр В. "Применение глубокого обучения для анализа рынка недвижимости." Недвижимость и строительство, 2019, стр. 23-30.
5. Montgomery, Douglas C., et al. "Introduction to Linear Regression Analysis." Wiley, 2012, стр. 58-102.
6. Бахвалов, Павел А. "Статистический анализ и моделирование данных с использованием R." Издательство МГУ, 2016, стр. 147-150.
7. Bishop, Christopher M. "Pattern Recognition and Machine Learning." Springer, 2006, стр. 140-145.
8. Hoerl, Arthur E., Kennard, Robert W. "Ridge Regression: Biased Estimation for Nonorthogonal Problems." Technometrics, 1970, стр. 55-67.
9. Tibshirani, Robert. "Regression Shrinkage and Selection via the Lasso." Journal of the Royal Statistical Society. Series B (Methodological), 1996, стр. 267-288.
10. Zou, Hui, Hastie, Trevor. "Regularization and Variable Selection via the Elastic Net." Journal of the Royal Statistical Society: Series B (Statistical Methodology), 2005, стр. 301-320.
11. Брейман, Лео. "Random Forests", Machine Learning, 2001.
12. Шмидт, Михаил. "Методы ансамблирования алгоритмов", Научно-технический журнал, 2018.
13. Фройнд, Йоав; Шапире, Роберт. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting", Journal of Computer and System Sciences, 1997.
14. Хасти, Тревор; Тибширани, Роберт; Фридман, Джером. "The Elements of Statistical Learning", Springer, 2009.
15. Хейст, Гарет, и др. "An Introduction to Statistical Learning", Springer, 2013.
16. Брейман, Лео. "Bagging predictors", Machine Learning, 1996.
17. Вольперт, Давид. "Stacked Generalization." Neural Networks, 1992, стр. 241-259.
18. Брейман, Лео. "Stacked Regressions." Machine Learning, 1996, стр. 49-64.
19. Зюко, Андрей Г. "Методы комбинирования классификаторов: стекинг и блендинг." Вестник СПбГУ, 2003, стр. 150-158.
20. Ткачев, Сергей. "Современные методы машинного обучения: учебное пособие." СПб.: БХВ-Петербург, 2018.
21. Боровский, Алексей. "Ансамблевые методы в машинном обучении." Москва: Физматлит, 2021, стр. 142-168.
22. Smith, John. "Advanced Machine Learning Techniques." New York: Springer, 2019, стр. 195-210.
23. Жуков, Павел. "Современные подходы к регрессионному анализу." Санкт-Петербург: БХВ-Петербург, 2020, стр. 88-102.
24. Чен, Тяньци, и Гю, Карлос. "XGBoost: A Scalable Tree Boosting System." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, стр. 785-794.
25. Кузнецов, Сергей. "Продвинутые методы ансамблирования в машинном обучении." Москва: Физматлит, 2018, стр. 132-160.
26. Прохоров, Александр. "Методы ансамблирования в машинном обучении." Москва: Наука, 2019, стр. 148-164.
27. Heaton, Jeff. "An Introduction to Neural Networks." Chicago: University of Chicago Press, 2017, стр. 81-102.
28. Hornik, Kurt. "Approximation Capabilities of Multilayer Feedforward Networks." Neural Networks, 1991, стр. 251-257
29. James, Gareth, Witten, Daniela, Hastie, Trevor, Tibshirani, Robert. "An Introduction to Statistical Learning." Springer, 2013.
30. Hastie, Trevor, Tibshirani, Robert, and Friedman, Jerome. "The Elements of Statistical Learning." Springer Series in Statistics, 2009, стр. 587-621.
31. Breiman, Leo. "Random Forests." Machine Learning, 2001, стр. 5-32
32. Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. "Deep Learning." MIT Press, 2016, стр. 215-240.
33. Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. "Deep Learning." MIT Press, 2016, стр. 215-240.
34. James, Gareth, et al. "An Introduction to Statistical Learning." Springer Texts in Statistics, 2013, стр. 442-466.
35. Kuhn, Max, и Johnson, Kjell. "Applied Predictive Modeling." Springer, 2013, стр. 228-244.
36. Козырев, Анатолий В. "Динамика региональных рынков недвижимости в современных условиях." Экономика и управление, 2019, стр. 88-95.
37. Smith, John. "Market Dynamics and Real Estate Valuation." Journal of Property Economics, 2018, стр. 204-219.
38. Ivanov, Dmitry, и Petrov, Alexey. "Challenges in Large-Scale Data Analysis." Computational Science Journal, 2020, стр. 142-156.
39. Taylor, Sarah E. "Deep Learning Overfitting: Causes and Solutions." AI Research, 2021, стр. 59-73.
40. Taylor, Sarah E. "Deep Learning Overfitting: Causes and Solutions." AI Research, 2021, стр. 59-73.
41. Taylor, Sarah E. "Deep Learning Overfitting: Causes and Solutions." AI Research, 2021, стр. 59-73.
42. Taylor, Sarah E. "Deep Learning Overfitting: Causes and Solutions." AI Research, 2021, стр. 59-73.

# 6. Приложение

## 6.1. Код и скрипты, использованные для обработки данных и разработки моделей.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import torch

import tensorflow as tf

from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler

from sklearn.impute import SimpleImputer, KNNImputer

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, GridSearchCV, RandomizedSearchCV, learning\_curve, KFold

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

from sklearn.linear\_model import LinearRegression, Ridge, Lasso, ElasticNet

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor, BaggingRegressor, StackingRegressor, VotingRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.pipeline import make\_pipeline

import xgboost as xgb

import lightgbm as lgb

import kerastuner as kt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, BatchNormalization

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.losses import MeanSquaredError

import optuna

import time

import warnings

from sklearn.metrics import make\_scorer

warnings.filterwarnings("ignore")

import random

new\_data = pd.read\_csv('kc\_house\_data.csv')

new\_data

new\_data = new\_data.drop(columns=['id', 'date'])

new\_data.shape

new\_data.describe()

new\_data.dtypes

new\_data.size

new\_data.isna().sum()

for column in new\_data.columns:

unique\_values\_count = new\_data[column].nunique()

print(f"Количество уникальных значений в столбце {column}: {unique\_values\_count}")

label\_encoder = LabelEncoder()

for column in new\_data.columns:

if (new\_data[column].dtype == 'object') or (new\_data[column].dtype == 'category'):

new\_data[column] = label\_encoder.fit\_transform(new\_data[column])

f,ax = plt.subplots(figsize=(12, 8))

sns.heatmap(new\_data.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)

plt.show()

new\_data = new\_data.drop(columns = ['sqft\_above', 'grade', 'bathrooms', 'sqft\_living15'])

new\_data

zero\_percentage = (new\_data[new\_data == 0].count() / len(new\_data)) \* 100

top\_zero\_columns = zero\_percentage.sort\_values(ascending=False)

print("Top columns with zero values and their percentages:")

for column, percentage in top\_zero\_columns.items():

print(f"{column}: {percentage:.2f}%")

selected\_columns = top\_zero\_columns.head(5).index.tolist()

selected\_data = new\_data[selected\_columns]

sns.pairplot(selected\_data, height=3)

plt.show()

new\_data = new\_data.drop(columns = ['yr\_renovated', 'sqft\_basement', 'waterfront'])

new\_data.columns

for column in ['price', 'bedrooms', 'sqft\_lot', 'floors', 'view',

'condition', 'yr\_built', 'zipcode', 'lat', 'long', 'sqft\_lot15']:

fig, ax = plt.subplots()

sns.boxplot(x=new\_data[column], ax=ax)

ax.set\_title(column)

plt.show()

outliers = new\_data[column][(new\_data[column] < new\_data[column].quantile(0.25) - 1.5 \* (new\_data[column].quantile(0.75) - new\_data[column].quantile(0.25))) | (new\_data[column] > new\_data[column].quantile(0.75) + 1.5 \* (new\_data[column].quantile(0.75) - new\_data[column].quantile(0.25)))]

print(f"Number of outliers in {column}: {len(outliers)}")

new\_data = new\_data.drop(outliers.index)

X = new\_data.drop('price', axis=1)

y = new\_data['price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf.fit(X\_train, y\_train)

feature\_importances = rf.feature\_importances\_

importance\_df = pd.DataFrame({'Feature': X\_train.columns, 'Importance': feature\_importances}).sort\_values(by='Importance', ascending=False)

importance\_df

new\_data = new\_data.drop(['view'], axis=1)

X = new\_data.drop('price', axis=1)

y = new\_data['price']

columns = X.columns

scaler = MinMaxScaler()

X\_scaled = scaler.fit\_transform(X)

X = pd.DataFrame(X\_scaled, columns=columns)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

y\_pred

linear\_r2 = r2\_score(y\_test, y\_pred)

linear\_mae = mean\_absolute\_error(y\_test, y\_pred)

linear\_mse = mean\_squared\_error(y\_test, y\_pred)

linear\_mape = np.mean(np.abs((y\_test - y\_pred) / y\_test)) \* 100

linear\_rmspe = np.sqrt(np.mean(np.square((y\_test - y\_pred) / y\_test))) \* 100

print("Linear\_R^2:", linear\_r2)

print("Linear\_MAE:", linear\_mae)

print("Linear\_MSE:", linear\_mse)

print("Linear\_MAPE:", linear\_mape)

print("Linear\_RMSPE:", linear\_rmspe)

initial\_mae = mean\_absolute\_error(y\_test, y\_pred)

initial\_mse = mean\_squared\_error(y\_test, y\_pred)

initial\_r2 = model.score(X\_train, y\_train)

initial\_mape = np.mean(np.abs((y\_test - y\_pred) / y\_test)) \* 100

initial\_rmspe = np.sqrt(np.mean(np.square((y\_test - y\_pred) / y\_test))) \* 100

print(f"Initial MAE: {initial\_mae}, Initial MSE: {initial\_mse}, Initial R2: {initial\_r2}, Initial MAPE: {initial\_mape}, Initial RMSPE: {initial\_rmspe}")

results = {}

for feature in X.columns:

X\_modified = X.drop(columns=[feature])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_modified, y, test\_size=0.2, random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

linear\_mae = mean\_absolute\_error(y\_test, y\_pred)

linear\_mse = mean\_squared\_error(y\_test, y\_pred)

linear\_r2 = r2\_score(y\_test, y\_pred)

linear\_mape = np.mean(np.abs((y\_test - y\_pred) / y\_test)) \* 100

linear\_rmspe = np.sqrt(np.mean(np.square((y\_test - y\_pred) / y\_test))) \* 100

results[feature] = (linear\_mae, linear\_mse, linear\_r2, linear\_mape, linear\_rmspe)

for feature, (linear\_mae, linear\_mse, linear\_r2, linear\_mape, linear\_rmspe) in results.items():

print(f"Feature: {feature}, MAE Change: {initial\_mae - linear\_mae}, MSE Change: {initial\_mse - linear\_mse}, R2 Change: {initial\_r2 - linear\_r2}, MAPE Change: {initial\_mape - linear\_mape}, RMSPE Change: {initial\_rmspe - linear\_rmspe}")

feature\_changes = []

for feature, (linear\_mae, linear\_mse, linear\_r2, linear\_mape, linear\_rmspe) in results.items():

mae\_change = initial\_mae - linear\_mae

mse\_change = initial\_mse - linear\_mse

r2\_change = initial\_r2 - linear\_r2

mape\_change = initial\_mape - linear\_mape

rmspe\_change = initial\_rmspe - linear\_rmspe

feature\_changes.append((feature, mae\_change, mse\_change, r2\_change, mape\_change, rmspe\_change))

top\_5\_mae = sorted(feature\_changes, key=lambda x: x[1], reverse=True)[:5]

top\_5\_mse = sorted(feature\_changes, key=lambda x: x[2], reverse=True)[:5]

top\_5\_r2 = sorted(feature\_changes, key=lambda x: x[3], reverse=True)[:5]

top\_5\_mape = sorted(feature\_changes, key=lambda x: x[4], reverse=True)[:5]

top\_5\_rmspe = sorted(feature\_changes, key=lambda x: x[5], reverse=True)[:5]

print("Top 5 features improving MAE:")

for feature, mae\_change, \_, \_, \_, \_ in top\_5\_mae:

print(f"Feature: {feature}, MAE Change: {mae\_change}")

print("\nTop 5 features improving MSE:")

for feature, \_, mse\_change, \_, \_, \_ in top\_5\_mse:

print(f"Feature: {feature}, MSE Change: {mse\_change}")

print("\nTop 5 features improving R2:")

for feature, \_, \_, r2\_change, \_, \_ in top\_5\_r2:

print(f"Feature: {feature}, R2 Change: {r2\_change}")

print("\nTop 5 features improving MAPE:")

for feature, \_, \_, \_, mape\_change, \_ in top\_5\_mape:

print(f"Feature: {feature}, MAPE Change: {mape\_change}")

print("\nTop 5 features improving RMSPE:")

for feature, \_, \_, \_, \_, rmspe\_change in top\_5\_rmspe:

print(f"Feature: {feature}, RMSPE Change: {rmspe\_change}")

new\_data = new\_data.drop(['long'], axis=1)

X = new\_data.drop('price', axis=1)

y = new\_data['price']

scaler = MinMaxScaler()

X\_scaled = scaler.fit\_transform(X)

X = pd.DataFrame(X\_scaled, columns=X.columns)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

y\_pred

corrected\_linear\_r2 = r2\_score(y\_test, y\_pred)

corrected\_linear\_mae = mean\_absolute\_error(y\_test, y\_pred)

corrected\_linear\_mse = mean\_squared\_error(y\_test, y\_pred)

corrected\_linear\_mape = np.mean(np.abs((y\_test - y\_pred) / y\_test)) \* 100

corrected\_linear\_rmspe = np.sqrt(np.mean(np.square((y\_test - y\_pred) / y\_test))) \* 100

print("Corrected\_Linear\_R^2:", corrected\_linear\_r2)

print("Corrected\_Linear\_MAE:", corrected\_linear\_mae)

print("Corrected\_Linear\_MSE:", corrected\_linear\_mse)

print("Corrected\_Linear\_MAPE:", corrected\_linear\_mape)

print("Corrected\_Linear\_RMSPE:", corrected\_linear\_rmspe)

def mean\_absolute\_percentage\_error(y\_true, y\_pred):

y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)

return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100

def root\_mean\_squared\_percentage\_error(y\_true, y\_pred):

y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)

return np.sqrt(np.mean(np.square((y\_true - y\_pred) / y\_true))) \* 100

basic\_lasso\_model = Lasso()

basic\_lasso\_model.fit(X\_train, y\_train)

basic\_lasso\_predictions = basic\_lasso\_model.predict(X\_test)

basic\_lasso\_mae = mean\_absolute\_error(y\_test, basic\_lasso\_predictions)

basic\_lasso\_mse = mean\_squared\_error(y\_test, basic\_lasso\_predictions)

basic\_lasso\_r2 = r2\_score(y\_test, basic\_lasso\_predictions)

basic\_lasso\_mape = mean\_absolute\_percentage\_error(y\_test, basic\_lasso\_predictions)

basic\_lasso\_rmspe = root\_mean\_squared\_percentage\_error(y\_test, basic\_lasso\_predictions)

print("Basic Lasso MAE:", basic\_lasso\_mae)

print("Basic Lasso MSE:", basic\_lasso\_mse)

print("Basic Lasso R-squared:", basic\_lasso\_r2)

print("Basic Lasso MAPE:", basic\_lasso\_mape)

print("Basic Lasso RMSPE:", basic\_lasso\_rmspe)

print("\n")

basic\_elasticnet\_model = ElasticNet()

basic\_elasticnet\_model.fit(X\_train, y\_train)

basic\_elasticnet\_predictions = basic\_elasticnet\_model.predict(X\_test)

basic\_elasticnet\_mae = mean\_absolute\_error(y\_test, basic\_elasticnet\_predictions)

basic\_elasticnet\_mse = mean\_squared\_error(y\_test, basic\_elasticnet\_predictions)

basic\_elasticnet\_r2 = r2\_score(y\_test, basic\_elasticnet\_predictions)

basic\_elasticnet\_mape = mean\_absolute\_percentage\_error(y\_test, basic\_elasticnet\_predictions)

basic\_elasticnet\_rmspe = root\_mean\_squared\_percentage\_error(y\_test, basic\_elasticnet\_predictions)

print("Basic ElasticNet MAE:", basic\_elasticnet\_mae)

print("Basic ElasticNet MSE:", basic\_elasticnet\_mse)

print("Basic ElasticNet R-squared:", basic\_elasticnet\_r2)

print("Basic ElasticNet MAPE:", basic\_elasticnet\_mape)

print("Basic ElasticNet RMSPE:", basic\_elasticnet\_rmspe)

print("\n")

basic\_ridge\_model = Ridge()

basic\_ridge\_model.fit(X\_train, y\_train)

basic\_ridge\_predictions = basic\_ridge\_model.predict(X\_test)

basic\_ridge\_mae = mean\_absolute\_error(y\_test, basic\_ridge\_predictions)

basic\_ridge\_mse = mean\_squared\_error(y\_test, basic\_ridge\_predictions)

basic\_ridge\_r2 = r2\_score(y\_test, basic\_ridge\_predictions)

basic\_ridge\_mape = mean\_absolute\_percentage\_error(y\_test, basic\_ridge\_predictions)

basic\_ridge\_rmspe = root\_mean\_squared\_percentage\_error(y\_test, basic\_ridge\_predictions)

print("Basic Ridge MAE:", basic\_ridge\_mae)

print("Basic Ridge MSE:", basic\_ridge\_mse)

print("Basic Ridge R-squared:", basic\_ridge\_r2)

print("Basic Ridge MAPE:", basic\_ridge\_mape)

print("Basic Ridge RMSPE:", basic\_ridge\_rmspe)

param\_grid\_lasso = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}

lasso\_grid = GridSearchCV(Lasso(), param\_grid\_lasso, cv=5, scoring='neg\_mean\_squared\_error')

lasso\_grid.fit(X\_train, y\_train)

optimal\_alpha\_lasso = lasso\_grid.best\_params\_['alpha']

optimal\_lasso\_model = Lasso(alpha=optimal\_alpha\_lasso)

optimal\_lasso\_model.fit(X\_train, y\_train)

optimal\_lasso\_predictions = optimal\_lasso\_model.predict(X\_test)

optimal\_lasso\_mae = mean\_absolute\_error(y\_test, optimal\_lasso\_predictions)

optimal\_lasso\_mse = mean\_squared\_error(y\_test, optimal\_lasso\_predictions)

optimal\_lasso\_r2 = r2\_score(y\_test, optimal\_lasso\_predictions)

optimal\_lasso\_mape = mean\_absolute\_percentage\_error(y\_test, optimal\_lasso\_predictions)

optimal\_lasso\_rmspe = root\_mean\_squared\_percentage\_error(y\_test, optimal\_lasso\_predictions)

print("Optimal Lasso MAE:", optimal\_lasso\_mae)

print("Optimal Lasso MSE:", optimal\_lasso\_mse)

print("Optimal Lasso R-squared:", optimal\_lasso\_r2)

print("Optimal Lasso MAPE:", optimal\_lasso\_mape)

print("Optimal Lasso RMSPE:", optimal\_lasso\_rmspe)

print("Best Lasso Alpha:", optimal\_alpha\_lasso)

print("\n")

param\_grid\_elasticnet = {

'alpha': [0.001, 0.01, 0.1, 1, 10, 100],

'l1\_ratio': [0.1, 0.3, 0.5, 0.7, 0.9]

}

elasticnet\_grid = GridSearchCV(ElasticNet(), param\_grid\_elasticnet, cv=5, scoring='neg\_mean\_squared\_error')

elasticnet\_grid.fit(X\_train, y\_train)

optimal\_alpha\_elasticnet = elasticnet\_grid.best\_params\_['alpha']

optimal\_l1\_ratio\_elasticnet = elasticnet\_grid.best\_params\_['l1\_ratio']

optimal\_elasticnet\_model = ElasticNet(alpha=optimal\_alpha\_elasticnet, l1\_ratio=optimal\_l1\_ratio\_elasticnet)

optimal\_elasticnet\_model.fit(X\_train, y\_train)

optimal\_elasticnet\_predictions = optimal\_elasticnet\_model.predict(X\_test)

optimal\_elasticnet\_mae = mean\_absolute\_error(y\_test, optimal\_elasticnet\_predictions)

optimal\_elasticnet\_mse = mean\_squared\_error(y\_test, optimal\_elasticnet\_predictions)

optimal\_elasticnet\_r2 = r2\_score(y\_test, optimal\_elasticnet\_predictions)

optimal\_elasticnet\_mape = mean\_absolute\_percentage\_error(y\_test, optimal\_elasticnet\_predictions)

optimal\_elasticnet\_rmspe = root\_mean\_squared\_percentage\_error(y\_test, optimal\_elasticnet\_predictions)

print("Optimal ElasticNet MAE:", optimal\_elasticnet\_mae)

print("Optimal ElasticNet MSE:", optimal\_elasticnet\_mse)

print("Optimal ElasticNet R-squared:", optimal\_elasticnet\_r2)

print("Optimal ElasticNet MAPE:", optimal\_elasticnet\_mape)

print("Optimal ElasticNet RMSPE:", optimal\_elasticnet\_rmspe)

print("Best ElasticNet Alpha:", optimal\_alpha\_elasticnet)

print("Best ElasticNet L1 Ratio:", optimal\_l1\_ratio\_elasticnet)

print("\n")

param\_grid\_ridge = {'alpha': [0.001, 0.01, 0.1, 1, 10, 100]}

ridge\_grid = GridSearchCV(Ridge(), param\_grid\_ridge, cv=5, scoring='neg\_mean\_squared\_error')

ridge\_grid.fit(X\_train, y\_train)

optimal\_alpha\_ridge = ridge\_grid.best\_params\_['alpha']

optimal\_ridge\_model = Ridge(alpha=optimal\_alpha\_ridge)

optimal\_ridge\_model.fit(X\_train, y\_train)

optimal\_ridge\_predictions = optimal\_ridge\_model.predict(X\_test)

optimal\_ridge\_mae = mean\_absolute\_error(y\_test, optimal\_ridge\_predictions)

optimal\_ridge\_mse = mean\_squared\_error(y\_test, optimal\_ridge\_predictions)

optimal\_ridge\_r2 = r2\_score(y\_test, optimal\_ridge\_predictions)

optimal\_ridge\_mape = mean\_absolute\_percentage\_error(y\_test, optimal\_ridge\_predictions)

optimal\_ridge\_rmspe = root\_mean\_squared\_percentage\_error(y\_test, optimal\_ridge\_predictions)

print("Optimal Ridge MAE:", optimal\_ridge\_mae)

print("Optimal Ridge MSE:", optimal\_ridge\_mse)

print("Optimal Ridge R-squared:", optimal\_ridge\_r2)

print("Optimal Ridge MAPE:", optimal\_ridge\_mape)

print("Optimal Ridge RMSPE:",optimal\_ridge\_rmspe)

print("Best Ridge Alpha:", optimal\_alpha\_ridge)

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, optimal\_ridge\_predictions, label='Ridge', alpha=0.5)

plt.scatter(y\_test, optimal\_lasso\_predictions, label='Lasso', alpha=0.5)

plt.scatter(y\_test, optimal\_elasticnet\_predictions, label='ElasticNet', alpha=0.5)

y\_test\_array = np.array(y\_test)

plt.plot(y\_test\_array, y\_test\_array, color='red', linestyle='--', linewidth=2, label='Ideal Predictions')

plt.xlabel('True Prices')

plt.ylabel('Predicted Prices')

plt.title('Regression Model Comparison')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

def root\_mean\_squared\_percentage\_error(y\_true, y\_pred):

return np.sqrt(np.mean(np.square((y\_true - y\_pred) / y\_true))) \* 100

rf\_model = RandomForestRegressor(n\_estimators=10, random\_state=42)

ab\_model = AdaBoostRegressor(DecisionTreeRegressor(max\_depth=3), n\_estimators=10, random\_state=42)

gb\_model = GradientBoostingRegressor(n\_estimators=10, random\_state=42)

bg\_model = BaggingRegressor(DecisionTreeRegressor(max\_depth=3), n\_estimators=10, random\_state=42)

stack\_model = StackingRegressor(estimators=[('rf', rf\_model), ('ab', ab\_model), ('gb', gb\_model)], final\_estimator=LinearRegression())

voting\_model = VotingRegressor(estimators=[('rf', rf\_model), ('ab', ab\_model), ('gb', gb\_model)])

models = [rf\_model, ab\_model, gb\_model, bg\_model, stack\_model, voting\_model]

# Создание scorer для RMSPE

rmspe\_scorer = make\_scorer(root\_mean\_squared\_percentage\_error, greater\_is\_better=False)

for model in models:

start\_time = time.time()

scores\_mae = cross\_val\_score(model, X, y, cv=5, scoring='neg\_mean\_absolute\_error')

scores\_mse = cross\_val\_score(model, X, y, cv=5, scoring='neg\_mean\_squared\_error')

scores\_r2 = cross\_val\_score(model, X, y, cv=5, scoring='r2')

scores\_mape = cross\_val\_score(model, X, y, cv=5, scoring='neg\_mean\_absolute\_percentage\_error')

scores\_rmspe = cross\_val\_score(model, X, y, cv=5, scoring=rmspe\_scorer)

mae = -scores\_mae.mean()

mse = -scores\_mse.mean()

r2 = scores\_r2.mean()

mape = -scores\_mape.mean()\*100

rmspe = -scores\_rmspe.mean()

model\_name = model.\_\_class\_\_.\_\_name\_\_

print(f'{model\_name}:\nMean absolute error: {mae} (+/- {scores\_mae.std() \* 2})')

print(f'Mean squared error: {mse} (+/- {scores\_mse.std() \* 2})')

print(f'R^2: {r2}')

print(f'Mean absolute percentage error: {mape} (+/- {scores\_mape.std() \* 2})')

print(f'Root mean squared percentage error: {rmspe} (+/- {scores\_rmspe.std() \* 2})')

print(f'time:{time.time() - start\_time}\n')

# Сохранение результатов для каждой модели

if model\_name == 'RandomForestRegressor':

rf\_model\_mae, rf\_model\_mse, rf\_model\_r2, rf\_model\_mape, rf\_model\_rmspe = mae, mse, r2, mape, rmspe

elif model\_name == 'AdaBoostRegressor':

ab\_model\_mae, ab\_model\_mse, ab\_model\_r2, ab\_model\_mape, ab\_model\_rmspe = mae, mse, r2, mape, rmspe

elif model\_name == 'GradientBoostingRegressor':

gb\_model\_mae, gb\_model\_mse, gb\_model\_r2, gb\_model\_mape, gb\_model\_rmspe = mae, mse, r2, mape, rmspe

elif model\_name == 'BaggingRegressor':

bg\_model\_mae, bg\_model\_mse, bg\_model\_r2, bg\_model\_mape, bg\_model\_rmspe = mae, mse, r2, mape, rmspe

elif model\_name == 'StackingRegressor':

stack\_model\_mae, stack\_model\_mse, stack\_model\_r2, stack\_model\_mape, stack\_model\_rmspe = mae, mse, r2, mape, rmspe

elif model\_name == 'VotingRegressor':

voting\_model\_mae, voting\_model\_mse, voting\_model\_r2, voting\_model\_mape, voting\_model\_rmspe = mae, mse, r2, mape, rmspe

def mean\_absolute\_percentage\_error(y\_true, y\_pred):

y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)

return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100

def root\_mean\_squared\_percentage\_error(y\_true, y\_pred):

return np.sqrt(np.mean(np.square((y\_true - y\_pred) / y\_true))) \* 100

rf\_model = RandomForestRegressor(n\_estimators=10, random\_state=42)

ab\_model = AdaBoostRegressor(DecisionTreeRegressor(max\_depth=3), n\_estimators=10, random\_state=42)

gb\_model = GradientBoostingRegressor(n\_estimators=10, random\_state=42)

estimators = [('rf', rf\_model), ('ab', ab\_model), ('gb', gb\_model)]

stack\_model = StackingRegressor(estimators=estimators, final\_estimator=LinearRegression())

params = {'rf\_\_n\_estimators': [50, 100], 'rf\_\_max\_depth': [5, 10],

'ab\_\_n\_estimators': [50, 100], 'ab\_\_learning\_rate': [0.1, 0.5],

'gb\_\_n\_estimators': [50, 100], 'gb\_\_max\_depth': [5, 10]}

pipeline = make\_pipeline(StandardScaler(with\_mean=False), LinearRegression())

rand\_search = RandomizedSearchCV(estimator=stack\_model, param\_distributions=params, cv=2, verbose=10)

rand\_search.fit(X, y)

best\_stack\_model = rand\_search.best\_estimator\_

stack\_model\_predictions = best\_stack\_model.predict(X\_test)

best\_stack\_model\_mae = mean\_absolute\_error(y\_test, stack\_model\_predictions)

best\_stack\_model\_mse = mean\_squared\_error(y\_test, stack\_model\_predictions)

best\_stack\_model\_r2 = r2\_score(y\_test, stack\_model\_predictions)

best\_stack\_model\_mape = mean\_absolute\_percentage\_error(y\_test, stack\_model\_predictions)

best\_stack\_model\_rmspe = root\_mean\_squared\_percentage\_error(y\_test, stack\_model\_predictions)

print("Best Stack Model MAE:", best\_stack\_model\_mae)

print("Best Stack Model MSE:", best\_stack\_model\_mse)

print("Best Stack Model R-squared:", best\_stack\_model\_r2)

print("Best Stack Model MAPE:", best\_stack\_model\_mape)

print("Best Stack Model RMSPE:", best\_stack\_model\_rmspe)

best\_hyperparameters = rand\_search.best\_params\_

print("Best Hyperparameters:", best\_hyperparameters)

import matplotlib.pyplot as plt

import numpy as np

from sklearn.model\_selection import learning\_curve

def plot\_and\_analyze\_learning\_curve(estimator, title, X, y, ylim=None, cv=None, n\_jobs=None, train\_sizes=np.linspace(.1, 1.0, 5)):

plt.figure()

plt.title(title)

if ylim is not None:

plt.ylim(\*ylim)

plt.xlabel("Training examples")

plt.ylabel("Score")

# Получение данных кривых обучения

train\_sizes, train\_scores, test\_scores = learning\_curve(

estimator, X, y, cv=cv, n\_jobs=n\_jobs, train\_sizes=train\_sizes,

scoring="neg\_mean\_squared\_error", verbose=10)

# Перевод в положительные значения ошибок

train\_scores\_mean = -np.mean(train\_scores, axis=1)

train\_scores\_std = np.std(train\_scores, axis=1)

test\_scores\_mean = -np.mean(test\_scores, axis=1)

test\_scores\_std = np.std(test\_scores, axis=1)

plt.grid()

plt.fill\_between(train\_sizes, train\_scores\_mean - train\_scores\_std,

train\_scores\_mean + train\_scores\_std, alpha=0.1, color="r")

plt.fill\_between(train\_sizes, test\_scores\_mean - test\_scores\_std,

test\_scores\_mean + test\_scores\_std, alpha=0.1, color="g")

plt.plot(train\_sizes, train\_scores\_mean, 'o-', color="r",

label="Training score")

plt.plot(train\_sizes, test\_scores\_mean, 'o-', color="g",

label="Cross-validation score")

plt.legend(loc="best")

# Пример использования этой функции:

title = "Learning Curves (Stacked Regressor)"

cv = 2 # Количество разделений для кросс-валидации

plot\_and\_analyze\_learning\_curve(best\_stack\_model, title, X, y, cv=cv, n\_jobs=-1)

xgb\_model = xgb.XGBRegressor(objective='reg:squarederror', colsample\_bytree=0.3, learning\_rate=0.1,

max\_depth=5, alpha=10, n\_estimators=100)

xgb\_model.fit(X\_train, y\_train)

xgb\_pred = xgb\_model.predict(X\_test)

xgb\_mae = mean\_absolute\_error(y\_test, xgb\_pred)

xgb\_mse = mean\_squared\_error(y\_test, xgb\_pred)

xgb\_r2 = r2\_score(y\_test, xgb\_pred)

xgb\_mape = mean\_absolute\_percentage\_error(y\_test, xgb\_pred)

xgb\_rmspe = root\_mean\_squared\_percentage\_error(y\_test, xgb\_pred)

print('XGBoost MAE:', xgb\_mae)

print('XGBoost MSE:', xgb\_mse)

print('XGBoost R^2:', xgb\_r2)

print('XGBoost MAPE:', xgb\_mape)

print('XGBoost RMSPE:', xgb\_rmspe)

lgbm\_model = lgb.LGBMRegressor()

lgbm\_model.fit(X\_train, y\_train)

lgbm\_pred = lgbm\_model.predict(X\_test)

lgbm\_mae = mean\_absolute\_error(y\_test, lgbm\_pred)

lgbm\_mse = mean\_squared\_error(y\_test, lgbm\_pred)

lgbm\_r2 = r2\_score(y\_test, lgbm\_pred)

lgbm\_mape = mean\_absolute\_percentage\_error(y\_test, lgbm\_pred)

lgbm\_rmspe = root\_mean\_squared\_percentage\_error(y\_test, lgbm\_pred)

print('LightGBM MAE:', lgbm\_mae)

print('LightGBM MSE:', lgbm\_mse)

print('LightGBM R^2:', lgbm\_r2)

print('LightGBM MAPE:', lgbm\_mape)

print('LightGBM RMSPE:', lgbm\_rmspe)

def xgb\_objective(trial):

param = {

'objective': 'reg:squarederror',

'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.1, 1),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3),

'max\_depth': trial.suggest\_int('max\_depth', 3, 10),

'alpha': trial.suggest\_float('alpha', 0, 10),

'n\_estimators': trial.suggest\_int('n\_estimators', 50, 300)

}

model = xgb.XGBRegressor(\*\*param)

model.fit(X\_train, y\_train)

preds = model.predict(X\_test)

rmse = mean\_squared\_error(y\_test, preds, squared=False)

return rmse

xgb\_study = optuna.create\_study(direction='minimize')

xgb\_study.optimize(xgb\_objective, n\_trials=50)

best\_xgb\_params = xgb\_study.best\_params

best\_xgb\_model = xgb.XGBRegressor(\*\*best\_xgb\_params)

best\_xgb\_model.fit(X\_train, y\_train)

best\_xgb\_preds = best\_xgb\_model.predict(X\_test)

best\_xgb\_mae = mean\_absolute\_error(y\_test, best\_xgb\_preds)

best\_xgb\_mse = mean\_squared\_error(y\_test, best\_xgb\_preds)

best\_xgb\_r2 = r2\_score(y\_test, best\_xgb\_preds)

best\_xgb\_mape = mean\_absolute\_percentage\_error(y\_test, best\_xgb\_preds)

best\_xgb\_rmspe = root\_mean\_squared\_percentage\_error(y\_test, best\_xgb\_preds)

print('Best XGBoost MAE:', best\_xgb\_mae)

print('Best XGBoost MSE:', best\_xgb\_mse)

print('Best XGBoost R^2:', best\_xgb\_r2)

print('Best XGBoost MAPE:', best\_xgb\_mape)

print('Best XGBoost RMSPE:', best\_xgb\_rmspe)

print('Best XGBoost Hyperparameters:', best\_xgb\_params)

def objective(trial):

param = {

'objective': 'regression',

'metric': 'rmse',

'verbosity': -1,

'boosting\_type': 'gbdt',

'num\_leaves': trial.suggest\_int('num\_leaves', 20, 100),

'max\_depth': trial.suggest\_int('max\_depth', 3, 10),

'learning\_rate': trial.suggest\_float('learning\_rate', 0.01, 0.3),

'n\_estimators': trial.suggest\_int('n\_estimators', 50, 300)

}

model = lgb.LGBMRegressor(\*\*param)

model.fit(X\_train, y\_train)

preds = model.predict(X\_test)

rmse = mean\_squared\_error(y\_test, preds, squared=False)

return rmse

study = optuna.create\_study(direction='minimize')

study.optimize(objective, n\_trials=50)

best\_params = study.best\_params

best\_lgbm\_model = lgb.LGBMRegressor(\*\*best\_params)

best\_lgbm\_model.fit(X\_train, y\_train)

best\_preds = best\_lgbm\_model.predict(X\_test)

best\_lgbm\_mae = mean\_absolute\_error(y\_test, best\_preds)

best\_lgbm\_mse = mean\_squared\_error(y\_test, best\_preds)

best\_lgbm\_r2 = r2\_score(y\_test, best\_preds)

best\_lgbm\_mape = mean\_absolute\_percentage\_error(y\_test, best\_preds)

best\_lgbm\_rmspe = root\_mean\_squared\_percentage\_error(y\_test, best\_preds)

print('Best LightGBM MAE:', best\_lgbm\_mae)

print('Best LightGBM MSE:', best\_lgbm\_mse)

print('Best LightGBM R^2:', best\_lgbm\_r2)

print('Best LightGBM MAPE:', best\_lgbm\_mape)

print('Best LightGBM RMSPE:', best\_lgbm\_rmspe)

print('Best LightGBM Hyperparameters:', best\_params)

def mean\_absolute\_percentage\_error(y\_true, y\_pred):

return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100

def root\_mean\_squared\_percentage\_error(y\_true, y\_pred):

return np.sqrt(np.mean(np.square((y\_true - y\_pred) / y\_true))) \* 100

def build\_and\_train\_model(X\_train, y\_train, X\_test, y\_test):

model = Sequential([

Dense(128, activation='relu', input\_dim=X\_train.shape[1]),

BatchNormalization(),

Dropout(0.2),

Dense(128, activation='relu'),

BatchNormalization(),

Dropout(0.2),

Dense(1, activation='linear')

])

model.compile(optimizer=Adam(0.001), loss='mse', metrics=['mean\_absolute\_error'])

model.fit(X\_train, y\_train, epochs=50, validation\_split=0.2, verbose=1)

predictions = model.predict(X\_test).flatten()

mae = mean\_absolute\_error(y\_test, predictions)

mse = mean\_squared\_error(y\_test, predictions)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, predictions)

mape = mean\_absolute\_percentage\_error(y\_test, predictions)

rmspe = root\_mean\_squared\_percentage\_error(y\_test, predictions)

return mae, mse, rmse, r2, mape, rmspe

initial\_mae, initial\_mse, initial\_rmse, initial\_r2, initial\_mape, initial\_rmspe = build\_and\_train\_model(X\_train, y\_train, X\_test, y\_test)

print(f'Initial Neural Network MAE: {initial\_mae}')

print(f'Initial Neural Network MSE: {initial\_mse}')

print(f'Initial Neural Network RMSE: {initial\_rmse}')

print(f'Initial Neural Network R^2: {initial\_r2}')

print(f'Initial Neural Network MAPE: {initial\_mape}')

print(f'Initial Neural Network RMSPE: {initial\_rmspe}')

def build\_model(hp):

model = Sequential()

model.add(Dense(units=hp.Int('units', min\_value=32, max\_value=512, step=32),

activation='relu', input\_dim=X\_train.shape[1]))

model.add(BatchNormalization())

model.add(Dropout(rate=hp.Float('dropout', min\_value=0.0, max\_value=0.5, step=0.1)))

model.add(Dense(units=hp.Int('units', min\_value=32, max\_value=512, step=32),

activation='relu'))

model.add(BatchNormalization())

model.add(Dropout(rate=hp.Float('dropout', min\_value=0.0, max\_value=0.5, step=0.1)))

model.add(Dense(1, activation='linear'))

model.compile(optimizer=Adam(hp.Float('learning\_rate', min\_value=1e-4, max\_value=1e-2, sampling='LOG')),

loss='mse', metrics=['mean\_absolute\_error'])

return model

tuner = kt.RandomSearch(

build\_model,

objective='val\_mean\_absolute\_error',

max\_trials=20,

executions\_per\_trial=1,

directory='tuning',

project\_name='keras\_tuning'

)

tuner.search(X\_train, y\_train, epochs=50, validation\_split=0.2, verbose=1)

best\_model = tuner.get\_best\_models(num\_models=1)[0]

best\_hyperparameters = tuner.get\_best\_hyperparameters()[0]

print(f'Best Hyperparameters: {best\_hyperparameters.values}')

predictions = best\_model.predict(X\_test).flatten()

best\_initial\_mae = mean\_absolute\_error(y\_test, predictions)

best\_initial\_mse = mean\_squared\_error(y\_test, predictions)

best\_initial\_rmse = np.sqrt(best\_initial\_mse)

best\_initial\_r2 = r2\_score(y\_test, predictions)

best\_initial\_mape = mean\_absolute\_percentage\_error(y\_test, predictions)

best\_initial\_rmspe = root\_mean\_squared\_percentage\_error(y\_test, predictions)

print(f'Best Neural Network MAE: {best\_initial\_mae}')

print(f'Best Neural Network MSE: {best\_initial\_mse}')

print(f'Best Neural Network RMSE: {best\_initial\_rmse}')

print(f'Best Neural Network R^2: {best\_initial\_r2}')

print(f'Best Neural Network MAPE: {best\_initial\_mape}')

print(f'Best Neural Network RMSPE: {best\_initial\_rmspe}')

models = ["Linear", "Corrected\_Linear", "Lasso", "Elasticnet", "Ridge", "Best\_Lasso", "Best\_Elasticnet", "Best\_Ridge", "RandomForestRegressor",

"AdaBoostRegressor", "GradientBoostingRegressor", "BaggingRegressor", "StackingRegressor",

"VotingRegressor", "Best\_StackingRegressor", "XGBRegressor", "LGBMRegressor", "Best\_LGBMRegressor", "Best\_XGBRegressor", "NN", "Best\_NN"]

mae\_values=[linear\_mae, corrected\_linear\_mae, basic\_lasso\_mae, basic\_elasticnet\_mae, basic\_ridge\_mae, optimal\_lasso\_mae, optimal\_elasticnet\_mae, optimal\_ridge\_mae, rf\_model\_mae, ab\_model\_mae, gb\_model\_mae, bg\_model\_mae, stack\_model\_mae, voting\_model\_mae, best\_stack\_model\_mae, xgb\_mae, lgbm\_mae, best\_xgb\_mae, best\_lgbm\_mae, initial\_mae, best\_initial\_mae]

mse\_values=[linear\_mse, corrected\_linear\_mse, basic\_lasso\_mse, basic\_elasticnet\_mse, basic\_ridge\_mse, optimal\_lasso\_mse, optimal\_elasticnet\_mse, optimal\_ridge\_mse, rf\_model\_mse, ab\_model\_mse, gb\_model\_mse, bg\_model\_mse, stack\_model\_mse, voting\_model\_mse, best\_stack\_model\_mse, xgb\_mse, lgbm\_mse, best\_xgb\_mse, best\_lgbm\_mse, initial\_mse, best\_initial\_mse]

r2\_values=[linear\_r2, corrected\_linear\_r2, basic\_lasso\_r2, basic\_elasticnet\_r2, basic\_ridge\_r2, optimal\_lasso\_r2, optimal\_elasticnet\_r2, optimal\_ridge\_r2, rf\_model\_r2, ab\_model\_r2, gb\_model\_r2, bg\_model\_r2, stack\_model\_r2, voting\_model\_r2, best\_stack\_model\_r2, xgb\_r2, lgbm\_r2, best\_xgb\_r2, best\_lgbm\_r2, initial\_r2, best\_initial\_r2]

mape\_values=[linear\_mape, corrected\_linear\_mape, basic\_lasso\_mape, basic\_elasticnet\_mape, basic\_ridge\_mape, optimal\_lasso\_mape, optimal\_elasticnet\_mape, optimal\_ridge\_mape, rf\_model\_mape, ab\_model\_mape, gb\_model\_mape, bg\_model\_mape, stack\_model\_mape, voting\_model\_mape, best\_stack\_model\_mape, xgb\_mape, lgbm\_mape, best\_xgb\_mape, best\_lgbm\_mape, initial\_mape, best\_initial\_mape]

rmspe\_values=[linear\_rmspe, corrected\_linear\_rmspe, basic\_lasso\_rmspe, basic\_elasticnet\_rmspe, basic\_ridge\_rmspe, optimal\_lasso\_rmspe, optimal\_elasticnet\_rmspe, optimal\_ridge\_rmspe, rf\_model\_rmspe, ab\_model\_rmspe, gb\_model\_rmspe, bg\_model\_rmspe, stack\_model\_rmspe, voting\_model\_rmspe, best\_stack\_model\_rmspe, xgb\_rmspe, lgbm\_rmspe, best\_xgb\_rmspe, best\_lgbm\_rmspe, initial\_rmspe, best\_initial\_rmspe]

def plot\_metric(values, metric\_name, color):

plt.figure(figsize=(16, 8))

bars = plt.bar(models, values, color=color)

plt.ylabel(metric\_name)

plt.title(f'{metric\_name} Across Models')

plt.xticks(rotation=45, ha='right')

plt.grid(axis='y', linestyle='--', alpha=0.7)

for bar in bars:

yval = bar.get\_height()

plt.text(bar.get\_x() + bar.get\_width() / 2, yval, round(yval, 2), ha='center', va='bottom', fontsize=10)

plt.tight\_layout()

plt.show()

plot\_metric(mae\_values, 'Mean Absolute Error (MAE)', 'skyblue')

plot\_metric(mse\_values, 'Mean Squared Error (MSE)', 'lightgreen')

plot\_metric(r2\_values, 'R^2', 'coral')

plot\_metric(mape\_values, 'Mean Absolute Percentage Error (MAPE)', 'orchid')

plot\_metric(rmspe\_values, 'Root Mean Squared Percentage Error (RMSPE)', 'gold')

random\_indices = random.sample(range(len(X\_test)), 10)

if isinstance(y\_test, pd.Series):

actual\_values = y\_test.iloc[random\_indices].values

else:

actual\_values = y\_test[random\_indices]

predicted\_values = stack\_model\_predictions[random\_indices]

differences = actual\_values - predicted\_values

data = {'Actual Values': actual\_values, 'Predicted Values': predicted\_values}

df = pd.DataFrame(data)

plt.figure(figsize=(12, 8))

bar\_width = 0.35

index = np.arange(10)

# Plot bars

bar1 = plt.bar(index, df['Actual Values'], bar\_width, label='Реальное')

bar2 = plt.bar(index + bar\_width, df['Predicted Values'], bar\_width, label='Прогнозное')

for i in range(len(differences)):

height = max(df['Actual Values'][i], df['Predicted Values'][i])

plt.text(index[i] + bar\_width/2, height, f'{differences[i]:.2f}', ha='center', va='bottom')

plt.xlabel('Номер примера')

plt.ylabel('Стоимость')

plt.title('Сравнение реальной и прогнозной стоимости')

plt.xticks(index + bar\_width / 2, [f"Пример {i+1}" for i in range(10)])

plt.legend()

plt.tight\_layout()

plt.show()