Изображение выглядит как эмблема, символ, герб, нашивка

Автоматически созданное описание

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА №4**

**по дисциплине**

**«Проектирование интеллектуальных систем (часть 1/2)»**

Студент группы:ИКБО-04-22 \_\_Кликушин В.И.\_ *(Ф. И.О. студента)*

Преподаватель \_\_ Холмогоров В.В.\_\_

*(Ф.И.О. преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2025

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc199496741)

[1 ПОСТАНОВКА ЗАДАЧИ 4](#_Toc199496742)

[2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 5](#_Toc199496743)

[2.1 Постановка задачи регрессии 5](#_Toc199496744)

[2.2 Метод наименьших квадратов 5](#_Toc199496745)

[2.3 Метрики регрессии 7](#_Toc199496746)

[2.4 Регрессия Лассо 9](#_Toc199496747)

[3 ДОКУМЕНТАЦИЯ К ДАННЫМ 10](#_Toc199496748)

[3.1 Описание предметной области 10](#_Toc199496749)

[3.2 Анализ данных 10](#_Toc199496750)

[3.3 Предобработка данных 14](#_Toc199496751)

[4 ПРАКТИЧЕСКАЯ ЧАСТЬ 17](#_Toc199496752)

[4.1 Метод наименьших квадратов 17](#_Toc199496753)

[4.2 Стохастический градиентный спуск 19](#_Toc199496754)

[4.3 Регрессия Лассо 20](#_Toc199496755)

[ЗАКЛЮЧЕНИЕ 22](#_Toc199496756)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 23](#_Toc199496757)

[ПРИЛОЖЕНИЯ 24](#_Toc199496758)

# ВВЕДЕНИЕ

Современные интеллектуальные аналитические системы всё чаще применяются для решения задач регрессии — прогнозирования непрерывных величин на основе набора признаков. Ранжирование цен, оценка финансовых показателей, прогнозирование результатов экзаменов и других образовательных метрик, оценка спроса на продукты и ресурсы, моделирование временных рядов – все эти прикладные задачи сводятся к построению регрессионной модели, способной минимизировать ошибку приближения между истинными и предсказанными значениями. Эффективность регрессии во многом зависит от качества предварительной обработки данных (удаления дубликатов, выбросов, масштабирования признаков), корректного кодирования категориальных переменных, устойчивости к мультиколлинеарности и сбалансированности выборки.

# 1 ПОСТАНОВКА ЗАДАЧИ

Цель работы: приобрести навыки применения моделей регрессии для решения прикладных задач анализа и сбора данных.

Задачи: определить предметную область решаемой задачи, найти или сгенерировать набор данных для выбранной задачи, проведя предварительную предобработку и подготовку данных, провести визуализацию предобработанного набора данных и выбрать для него модель регрессии, выбрать метрику ошибку и обучить модель регрессии, выполнить задачу предметной области, интерпретировать и визуализировать результаты обучения модели.

# 2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## 2.1 Постановка задачи регрессии

Пусть имеется обучающая выборка из наблюдений (Формула 2.1.1).

, (2.1.1)

где – вектор из признаков для -ого объекта,

– значение целевой переменной для -ого объекта.

Введем «искусственный» признак , чтобы учесть свободный член (Формула 2.1.2).

(2.1.2)

Вектор весов модели обозначим в соответствие с Формулой 2.1.3.

(2.1.3)

Линейная модель предсказания тогда задается Формулой 2.1.4.

(2.1.4)

Задача состоит в том, чтобы подобрать , минимизирующий расхождение между реальными и предсказанными .

## 2.2 Метод наименьших квадратов

Определим ошибку для -ого объекта (Формула 2.2.1).

(2.2.1)

Метод наименьших квадратов минимизирует сумму квадратов этих ошибок (Формула 2.2.2).

(2.2.2)

Формулируется следующая оптимизационная задача (Формула 2.2.3).

(2.2.3)

Построим регрессионную матрицу , в которой в -ой строке лежит вектор (Формула 2.2.4).

(2.2.4)

Вектор истинных ответов задается по Формуле 2.2.5.

(2.2.5)

Все предсказания можно записать в матричном виде (Формула 2.2.6).

(2.2.6)

Вектор ошибок в матричном виде задается по Формуле 2.2.7.

(2.2.7)

Функцию потерь можно переписать (Формула 2.2.8).

(2.2.8)

Вычислим градиент и приравняем его к нулю (Формула 2.2.9).

(2.2.9)

Это и есть нормальные уравнения. Если матрица невырождена (обратима), решение записывается явным образом по Формуле 2.2.10.

(2.2.10)

Вектор содержит оценки свободного члена и коэффициентов , минимизирующих сумму квадратичных ошибок.

## 2.3 Метрики регрессии

В задачах регрессии принято оценивать, насколько точно модель предсказывает числовые значения. Наиболее популярны следующие метрики:

1. **Mean Squared Error (MSE)**

MSE измеряет среднее арифметическое квадратов ошибок (остатков) модели. Он чувствителен к большим отклонениям, потому что разность возводится в квадрат. Если есть крупные выбросы (несоответствия), они сильно увеличивают MSE. Низкое значение MSE означает, что в среднем модель предсказывает близко к истинным (Формула 2.3.1).

, (2.3.1)

где – истинное значение целевой переменной для -ого объекта;

– предсказанное моделью значение для того же -ого объекта;

– число наблюдений в выборке.

1. **Root Mean Squared Error (RMSE)**

RMSE — это квадратный корень из MSE. Благодаря взятию корня, RMSE имеет те же единицы измерения, что и целевая переменная, и интерпретируется как средний «разброс» ошибки вокруг нуля. Чем меньше RMSE, тем точнее модель в абсолютных единицах. Вычисляется по Формуле 2.3.2.

(2.3.2)

1. **Mean Absolute Error (MAE)**

MAE рассчитывает среднее абсолютное значение ошибок. В отличие от MSE, здесь не происходит квадрирования, поэтому выбросы (редкие, но большие ошибки) не «штрафуются» так сильно. MAE показывает, насколько в среднем предсказание отличается от истинного значения по модулю. Метрика рассчитывается по Формуле 2.3.3.

(2.3.3)

1. **Median Absolute Error (MdAE)**

MdAE (медианная абсолютная ошибка) берёт медиану набора абсолютных ошибок. Это более робастная метрика, чем MAE, так как игнорирует половину экстремальных значений (самые большие превышения). Если в данных есть сильные выбросы, MdAE покажет, каким является «типичное» (медианное) абсолютное отклонение (Формула 2.3.4).

(2.3.4)

1. **R2 Score (коэффициент детерминации)**

Метрика показывает, какую долю разброса целевой переменной объясняет модель (Формула 2.3.5).

(2.3.5)

## 2.4 Регрессия Лассо

Цель регрессии Лассо — подобрать , минимизирующий не только квадратичную ошибку, но и штраф за сумму абсолютных значений весов .

Лассо добавляет к функции суммы квадратов ошибок пороговый штраф на все веса, кроме свободного члена . Параметр регуляризации обозначается через (Формула 2.4.1).

(2.4.1)

# 3 ДОКУМЕНТАЦИЯ К ДАННЫМ

## 3.1 Описание предметной области

В качестве набора данных выбран Student Performance Dataset, синтетический датасет, предназначенный для исследования факторов, влияющих на академическую успеваемость школьников. Набор состоит из 10 000 записей, каждая из которых содержит информацию о совокупности предикторов: количестве часов, потраченных на обучение (Hours Studied), предыдущих оценках (Previous Scores), участии во внеклассных мероприятиях (Extracurricular Activities), среднем количестве часов сна в сутки (Sleep Hours) и числе прорешанных контрольных или тренировочных вариантов (Sample Question Papers Practiced). В качестве целевой переменной выступает Performance Index – показатель общей успеваемости, округлённый до целого значения в диапазоне от 10 до 100 (чем выше индекс, тем лучше академический результат). Цель сбора и использования этих данных — проанализировать взаимосвязь между объёмом самостоятельной работы, уровнем прошлых достижений, участием в активностях вне уроков, режимом сна и практикой решения заданий с итоговым Performance Index. Результаты исследования могут помочь выявить ключевые детерминанты успешности учащихся, спрогнозировать академическую успеваемость и разработать рекомендации для оптимизации учебного процесса.

## 3.2 Анализ данных

В исходных данных представлено 10 000 записей об обучении студентов. Каждый объект соответствует одному учащемуся и содержит пять предикторных переменных и одну целевую переменную:

* Hours Studied (Часы обучения): общее количество часов, которые студент посвятил самостоятельной подготовке;
* Previous Scores (Предыдущие оценки): средний балл, полученный студентом на предыдущих контрольных работах или экзаменах;
* Extracurricular Activities (Внеклассные мероприятия): бинарный индикатор участия в дополнительной активности («Yes»/«No»);
* Sleep Hours (Часы сна): среднее число часов сна в сутки;
* Sample Question Papers Practiced (Прорешанные примерные варианты): количество тренировочных заданий, прорешанных студентом перед экзаменом;
* Performance Index (Индекс успеваемости): итоговый показатель, отражающий общий академический результат студента. Значение индекса округлено до целого и лежит в диапазоне от 10 до 100 (чем выше, тем лучше успеваемость).

Для предобработки и анализа датасета в файле dataset\_manager.py написан класс DatasetManager. Содержание файла dataset\_manager.py представлено в Приложении А.

Описательная статистика признаков отображена на Рисунке 3.2.1.

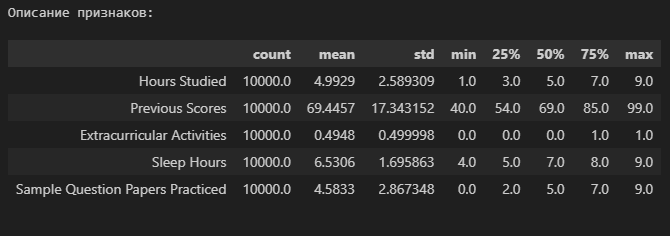


Рисунок 3.2.1 – Описательная статистика признаков

Распределение часов обучения (Hours Studied) демонстрирует относительную симметричность со средним значением 4.99 часа и медианой 5.0 часа, при этом наблюдается умеренная вариативность (стандартное отклонение 2.59) в диапазоне от 1 до 9 часов. Четверть студентов уделяет обучению не более 3 часов, тогда как 75% учатся до 7 часов ежедневно.

Показатель Previous Scores выделяется наиболее широким разбросом данных (стандартное отклонение 17.34) при среднем значении 69.45 баллов. Особого внимания требует нижний квартиль: 25% студентов имеют баллы не выше 54 при абсолютном минимуме 40 баллов, что указывает на возможное наличие выбросов в этой области. Верхние 25% учащихся демонстрируют результаты от 85 баллов.

Бинарный признак Extracurricular Activities показывает сбалансированное распределение: медиана 0.0 при среднем 0.495 свидетельствует, что примерно половина студентов вовлечена во внеклассную активность. Распределение часов сна (Sleep Hours) вызывает определенные опасения, так как четверть студентов спит не более 5 часов при медиане 7 часов и среднем значении 6.53 часа. Разброс значений от 4 до 9 часов с умеренным стандартным отклонением 1.7 указывает на потенциальную проблему дефицита сна у значительной части учащихся.

Признак Sample Question Papers Practiced характеризуется средним значением 4.58 при медиане 5.0 и значительном разбросе от 0 до 9 единиц (стандартное отклонение 2.87). Критичным представляется тот факт, что 25% студентов решают не более 2 пробных тестов, что может негативно влиять на их академические результаты. Полученные данные подчеркивают необходимость масштабирования числовых признаков, особенно Previous Scores, и проверки на выбросы в нижних квартилях.

На Рисунке 3.2.2 показано гистограммное распределение целевой переменной Performance Index.

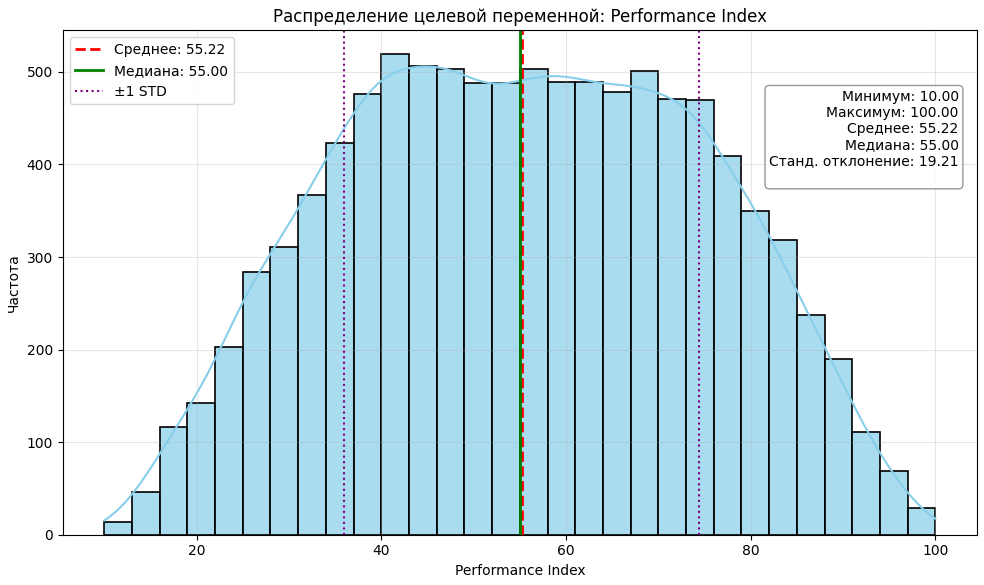


Рисунок 3.2.2 – Гистограмма распределения Performance Index

На Рисунке 3.2.2 видно, что есть небольшая доля экстремально низких и экстремально высоких значений индекса, однако большинство лежит между 60 и 90.

На Рисунке 3.2.3 представлены гистограммы распределений всех пяти признаков.

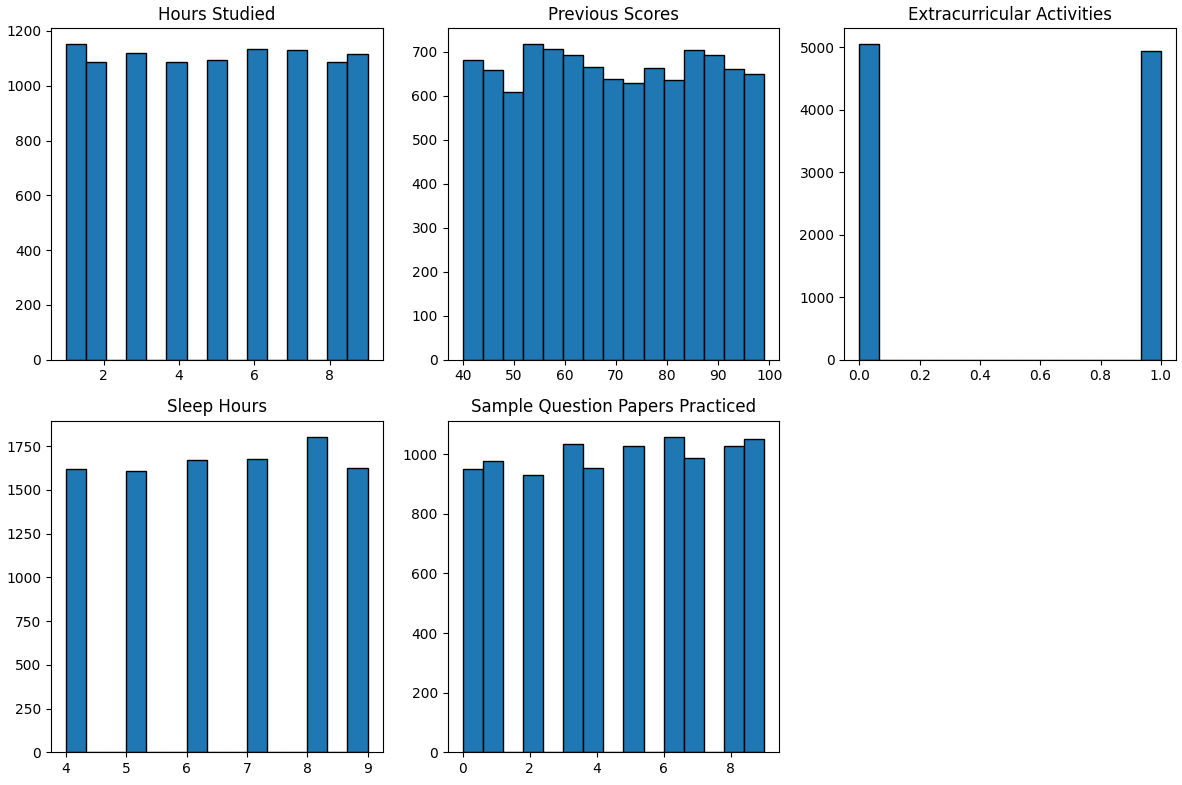


Рисунок 3.2.3 - Гистограммы распределений признаков

Признаки имеют нормальное распределение.

Матрица корреляции признаков представлена на Рисунке 3.2.4.

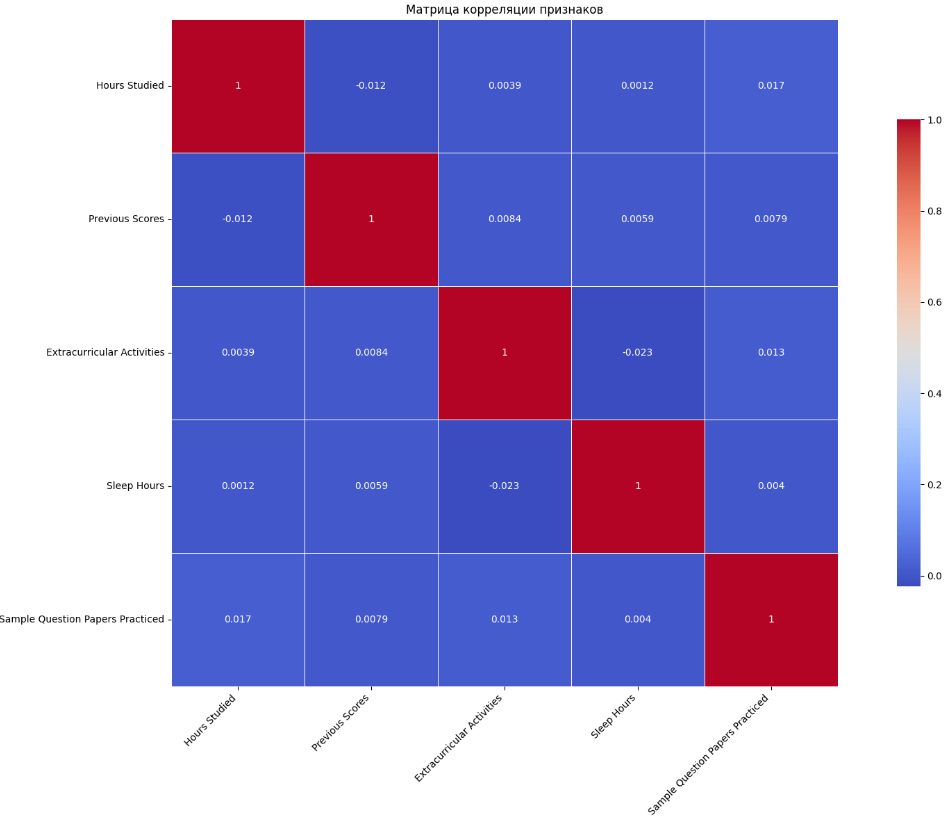


Рисунок 3.2.4 – Матрица корреляции признаков

Матрица указывает на сложную, нелинейную природу образовательных данных, где традиционные факторы успеваемости проявляют неожиданно слабые взаимосвязи. Высоких корреляций между признаками не имеется.

## 3.3 Предобработка данных

Реализованы следующие этапы предобработки данных:

* удаление дубликатов строк;
* удаление выбросов по Z-оценке;
* масштабирование признаков (StandardScaler);
* выбор и удаление избыточных признаков.

Удаление дубликатов строк подразумевает обнаружение и удаление полностью идентичных записей (строк) в исходном датасете. Под «идентичностью» понимается совпадение всех значений по всем признакам. В рассматриваемом датасете удалено 127 дубликатов.

Выбросы (аномальные значения) — это отдельные объекты, сильно отклоняющиеся от общей «массы» точек. Чаще всего они встречаются в признаках с широким диапазоном. Один из способов формального выявления выбросов — использовать Z-оценку.

Для каждого значения признака в образце рассчитывается величина по Формуле 3.3.1.

, (3.3.1)

где – среднее отклонение признака по всем образцам;

– стандартное отклонение признака по всем образцам.

Образец считается выбросом, если хотя бы один признак имеет .

В качестве порогового значения выбран = 3. Таким образом, удалены все образцы, в которых хотя бы один признак отклонён от среднего более чем на три стандартных отклонения. В результате удаления выбросов по Z-оценке было исключено десять строк из исходного набора данных.

Скалирование признаков — это приведение всех измеряемых величин в единый единичный масштаб. Без масштабирования признаки с большим диапазоном «будут весить» значительно больше при классификации, чем признаки с узким диапазоном.

StandardScaler — один из наиболее распространённых способов стандартизации. Для каждого признака вычисляется среднее и стандартное отклонение . Каждое значение преобразуется по Формуле 3.3.2.

, (3.3.2)

В результате стандартизированный признак ​ имеет среднее 0 и стандартное отклонение 1.

Данные разделены на обучающую и тестовую выборки в соотношении 80:20. Обучающая выборка содержит 7898 образца, а тестовая выборка – 1975 образцов.

# 4 ПРАКТИЧЕСКАЯ ЧАСТЬ

## 4.1 Метод наименьших квадратов

Для решения задачи регрессии использована библиотечная реализация метода наименьших квадратов (класс LinearRegression из scikit-learn). Модель обучена на предобработанных данных с последующей оценкой качества. Алгоритм написан в файле linear\_regression.py, содержание которого представлено в Приложении Б.

Построен график фактических меток и предсказанных для визуализации оценки качества модели (Рисунок 4.1.1).

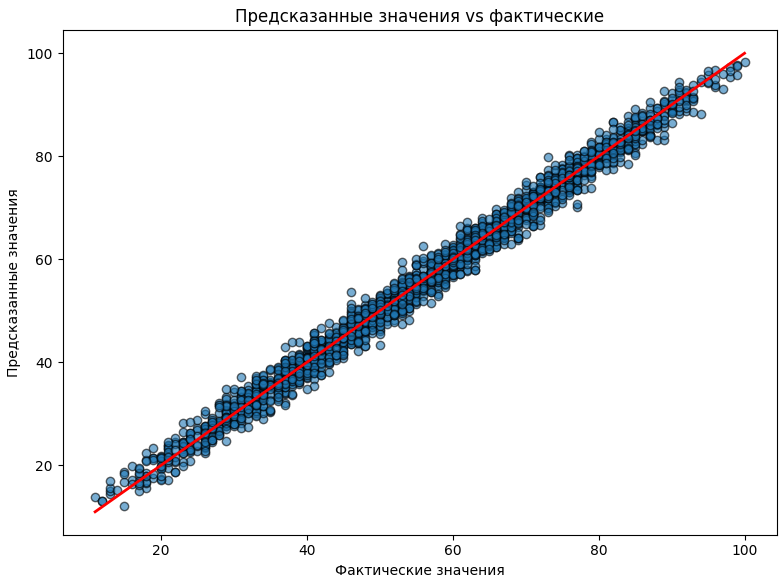


Рисунок 4.1.1 - График фактических меток и предсказанных для библиотечной реализации метода наименьших квадратов

Большинство точек аккуратно стянуты вдоль диагонали. Это говорит о том, что предсказания модели в среднем попадают близко к истинным .

Анализ коэффициентов представлен на Рисунке 4.1.2.

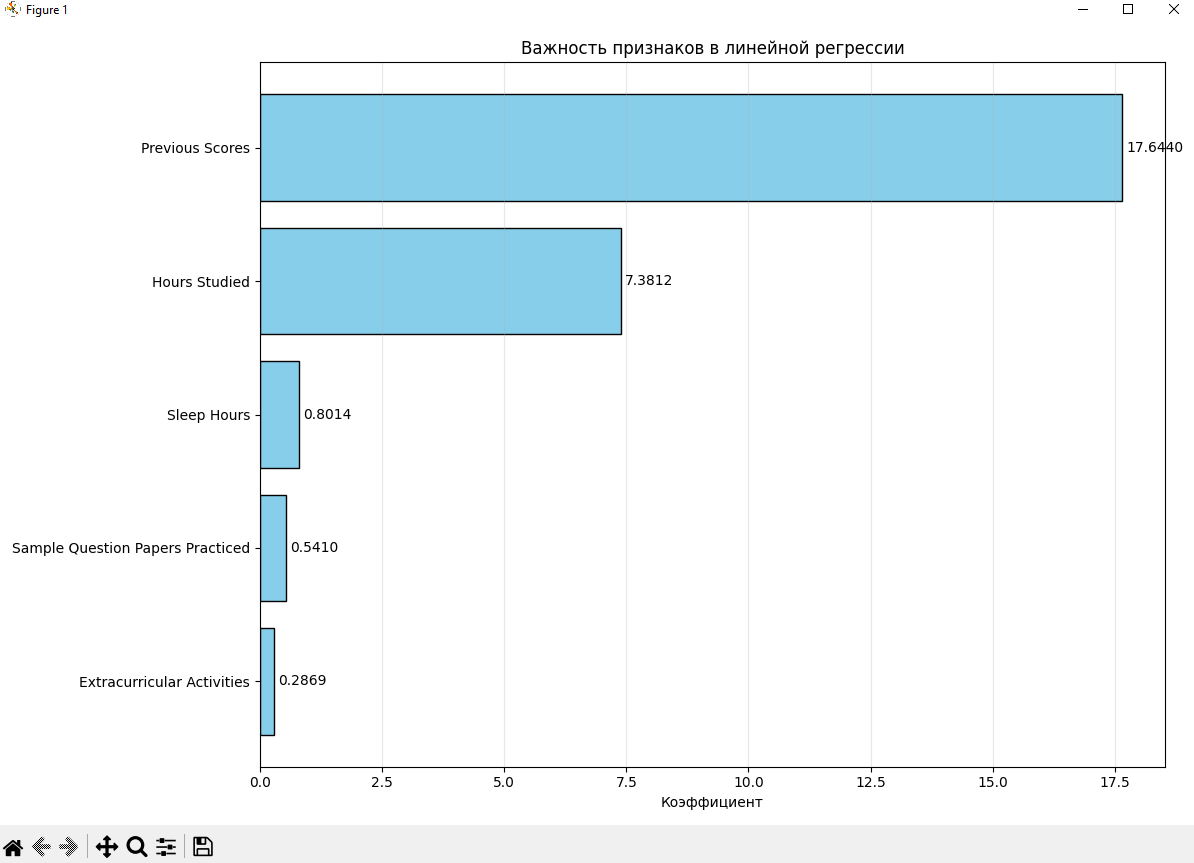


Рисунок 4.1.2 – Важность признаков для библиотечной реализации метода наименьших квадратов

Как и ожидалось, наиболее значимым коэффициентом является Previous Scores (Предыдущие баллы).

После обучения модели на обучающих данных, сделаны предсказания на тестовой выборке. По результатам вычислены метрики (Таблица 4.1.1).

Таблица 4.1.1 – Метрики по результатам классификации с библиотечной реализацией метода наименьших квадратов

|  |  |  |
| --- | --- | --- |
| Метрика | Значение | Интерпретация |
| MSE | 4.31 | В среднем квадрат отклонения модели от реального значения равен 4.31 |
| RMSE | 2.08 | Модель в среднем «промахивается» на ≈ 2.08 единицы |
| MAE | 1.65 | В среднем ошибка равна 1.65 |
| MDAE | 1.40 | Половина объектов прогнозируется с абсолютной ошибкой ≤ 1.4 |
| R2 | 0.99 | Модель объясняет 99 % дисперсии целевой переменной |

Результаты указывают на очень высокое качество модели.

## 4.2 Стохастический градиентный спуск

Разработана собственная реализация обучения линейной многомерной регрессии с помощью стохастического градиентного спуска. Алгоритм написан в файле linear\_regression\_custom.py, содержание которого представлено в Приложении В.

Кривая обучения представлена на Рисунке 4.2.1.

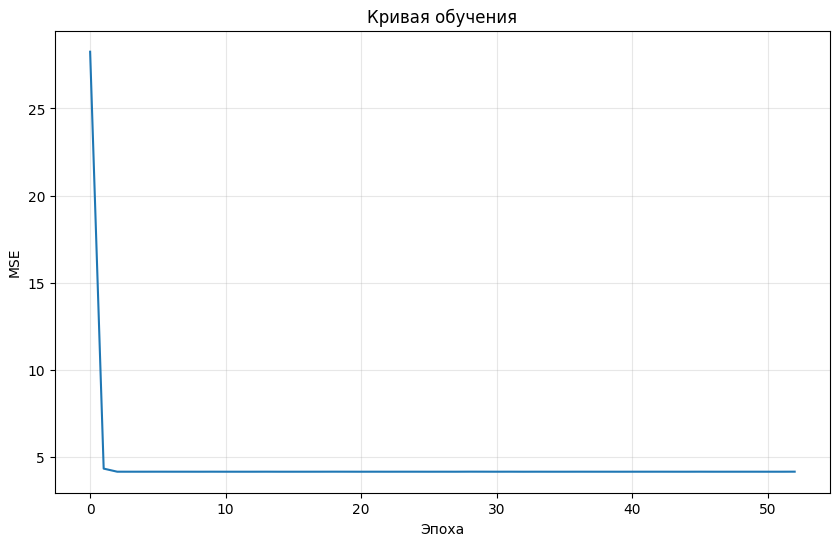


Рисунок 4.2.1 – Кривая обучения модели

В результате тестирования получены следующие метрики (Рисунок 4.2.2).

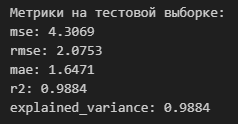


Рисунок 4.2.2 – Полученные метрики для собственной реализации стохастического градиентного спуска

Метрики сопоставимы с результатами библиотечной реализации метода наименьших квадратов, что говорит о правильности реализации алгоритма.

Значимость признаков отображена на Рисунке 4.2.3.

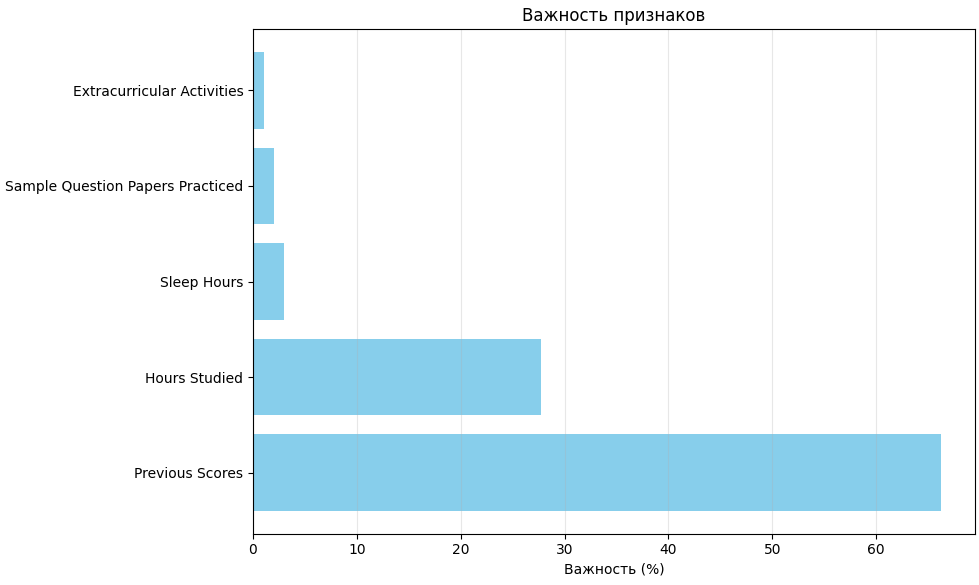


Рисунок 4.2.3 – Абсолютная значимость признаков в процентном соотношении

Наиболее значимыми остаются факторы, выявленные для библиотечной реализации метода наименьших квадратов.

## 4.3 Регрессия Лассо

Реализована регрессия Лассо в файле Lasso\_regression.py, содержание которого представлено в Приложении Г.

На Рисунке 4.3.1 представлены полученные значения коэффициентов.

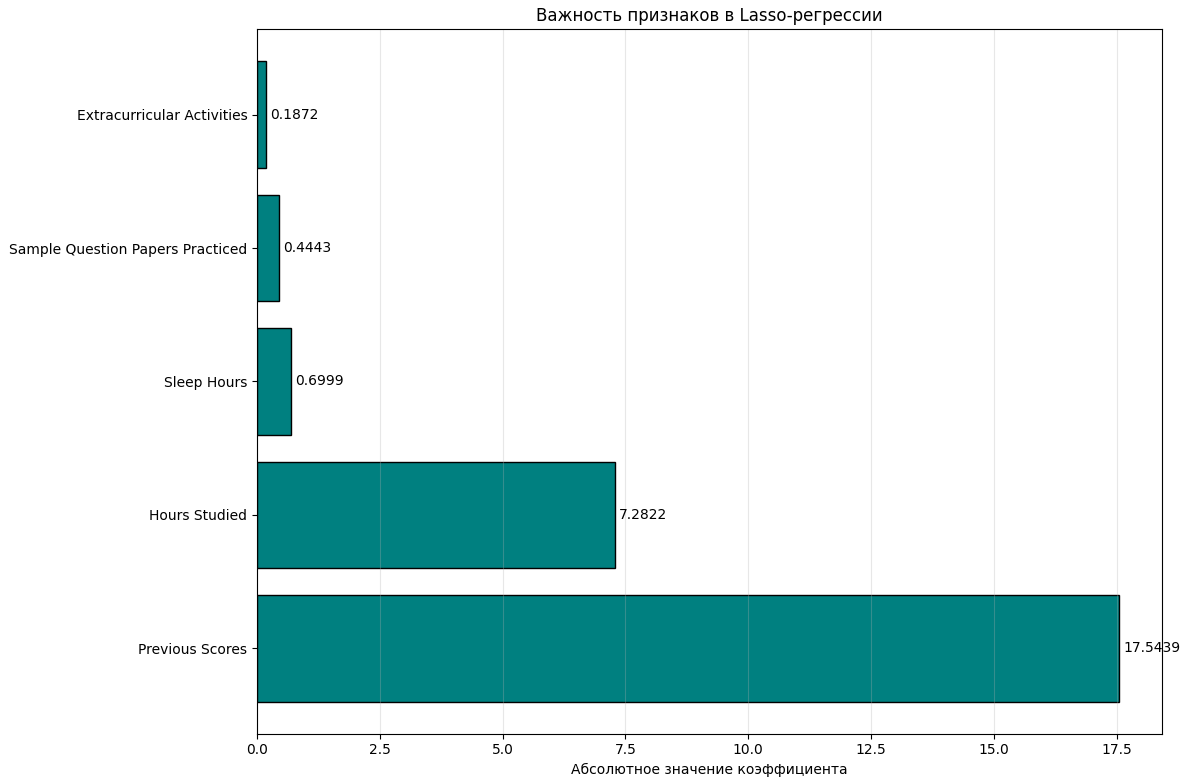


Рисунок 4.3.1 – Значения коэффициентов для регрессии Лассо

Так как между признаками нет сильных корреляций, L1 штраф лишь незначительно уменьшил значения коэффициентов.

# ЗАКЛЮЧЕНИЕ

В работе рассмотрены и применены три метода регрессии. Метод наименьших квадратов продемонстрировал высокую точность с коэффициентом детерминации R2=0.99, что свидетельствует о практически полном объяснении дисперсии целевой переменной.

Стохастический градиентный спуск реализован вручную и показал результаты, сопоставимые с библиотечной реализацией OLS, что подтвердило корректность алгоритма.

Регрессия Лассо использована для регуляризации модели. Добавление L1штрафа почти не улучшило результат, потому что нет сильных корреляций между признаками и каждый признак вносит свою уникальную долю объясняемой дисперсии.

# СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Сорокин, А. Б. Безусловная оптимизация. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк — М. РТУ МИРЭА , 2020.
2. Сорокин, А. Б. Введение в генетические алгоритмы: теория, расчеты и приложения. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин — М. МИРЭА , 2018.
3. Student Performance (Multiple Linear Regression) [Электронный ресурс]: Kaggle. URL: https://www.kaggle.com/datasets/nikhil7280/student-performance-multiple-linear-regression/data (Дата обращения: 19.05.2025).
4. Основы линейной регрессии [Электронный ресурс]: Habr. URL: https://habr.com/ru/articles/514818/ (Дата обращения: 20.05.2025).

# ПРИЛОЖЕНИЯ

Приложение А — Файл dataset\_manager.py для предобработки и анализа датасета.

Приложение Б — Файл linear\_regression.py с использованием готовой реализации алгоритма наименьших квадратов.

Приложение В — Файл linear\_regression\_custom.py с собственной реализацией стохастического градиентного спуска.

Приложение Г — Файл Lasso\_regression.py с использованием готовой реализацией регрессии Лассо.

### Приложение А

Файл dataset\_manager.py для предобработки и анализа датасета

Листинг А – Содержание файла dataset\_manager.py

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from typing import Optional, Dict, Tuple

from pandas import DataFrame, Series

class DatasetManager:

    """

    Менеджер датасета для задач регрессии: загрузка из CSV, анализ, предобработка и визуализация.

    Параметры:

        csv\_path (str): Путь к CSV-файлу с данными.

        target\_column (str): Название столбца с целевой переменной.

    Атрибуты:

        df (DataFrame): Полный исходный DataFrame.

        features (DataFrame): DataFrame с только признаками (без целевой переменной).

        target (Series): Серия с целевой переменной.

        scaled\_features (DataFrame): DataFrame с масштабированными признаками.

        stats (Dict[str, DataFrame]): Словарь рассчитанных статистик.

    """

    def \_\_init\_\_(self, csv\_path: str, target\_column: str) -> None:

        self.csv\_path: str = csv\_path

        self.target\_column: str = target\_column

        self.df: Optional[DataFrame] = None

        self.features: Optional[DataFrame] = None

        self.target: Optional[Series] = None

        self.scaled\_features: Optional[DataFrame] = None

        self.stats: Dict[str, DataFrame] = {}

        self.\_load\_data()

        self.\_dataset\_custom\_preprocess()

        self.\_extract\_features\_target()

        if self.csv\_path == 'housing\_dataset.csv':

            self.remove\_feature('Location')

            self.remove\_feature('City')

    def \_dataset\_custom\_preprocess(self):

        if self.csv\_path == 'Student\_Performance.csv':

            self.df["Extracurricular Activities"] = self.df["Extracurricular Activities"].map({"Yes":1 , "No":0})

        else:

            cols\_to\_check = [col for col in self.df.columns if col != "No. of Bedrooms"]

            mask\_no\_nines = (self.df[cols\_to\_check] != 9).all(axis=1)

            self.df = self.df[mask\_no\_nines].reset\_index(drop=True)

    def \_load\_data(self) -> None:

        """

        Загружает данные из CSV-файла в self.df.

Продолжение Листинга А

        Выбрасывает:

            FileNotFoundError: если файл по пути csv\_path не найден.

            pd.errors.EmptyDataError: если CSV пустой или неверный формат.

        """

        try:

            self.df = pd.read\_csv(self.csv\_path)

        except FileNotFoundError as e:

            raise FileNotFoundError(f"Файл не найден по пути: {self.csv\_path}") from e

        except pd.errors.EmptyDataError as e:

            raise pd.errors.EmptyDataError(

                f"Пустой или некорректный CSV: {self.csv\_path}"

            ) from e

        print(

            f"Данные загружены: {self.df.shape[0]} строк, {self.df.shape[1]} столбцов"

        )

    def \_extract\_features\_target(self) -> None:

        """

        Разделяет DataFrame на признаки и целевую переменную.

        После выполнения:

            - self.features содержит все столбцы, кроме target\_column.

            - self.target содержит Series с данными целевой переменной.

        Выбрасывает:

            ValueError: если target\_column отсутствует в self.df.

        """

        if self.df is None:

            raise RuntimeError("Данные не загружены. Сначала вызовите \_load\_data().")

        if self.target\_column not in self.df.columns:

            raise ValueError(

                f"Целевая переменная '{self.target\_column}' не найдена в данных."

            )

        self.target = self.df[self.target\_column].copy()

        self.features = self.df.drop(columns=[self.target\_column]).copy()

    def compute\_basic\_statistics(self) -> Dict[str, DataFrame]:

        """

        Вычисляет базовые статистики по признакам и сохраняет их в self.stats.

        Сохраняются:

            - "describe": описательные статистики (mean, std, min, max, квартили) для каждого признака.

            - "correlation\_matrix": матрица корреляций между признаками.

            - "target\_distribution": описательные статистики целевой переменной.

        Возвращает:

            Dict[str, DataFrame]: Словарь со статистиками.

        Выбрасывает:

            RuntimeError: если признаки или целевая переменная не выделены.

        """

        if self.features is None or self.target is None:

            raise RuntimeError(

                "Признаки или целевая переменная не выделены. Вызовите \_extract\_features\_target()."

            )

Продолжение Листинга А

        desc\_features = self.features.describe().T

        self.stats["describe"] = desc\_features

        corr\_features = self.features.corr()

        self.stats["correlation\_matrix"] = corr\_features

        desc\_target = self.target.describe().to\_frame(name="target\_stats")

        self.stats["target\_distribution"] = desc\_target

        return self.stats

    def preprocess(

        self,

        drop\_duplicates: bool = True,

        drop\_outliers: bool = True,

        z\_thresh: float = 3.0,

    ) -> None:

        """

        Предобработка данных:

            1. Удаление дубликатов (если drop\_duplicates=True).

            2. Удаление выбросов по Z-оценке (если drop\_outliers=True).

            3. Масштабирование признаков StandardScaler.

        Параметры:

            drop\_duplicates (bool): Удалять ли полные дубликаты строк.

            drop\_outliers (bool): Удалять ли выбросы по Z-оценке для каждого признака.

            z\_thresh (float): Порог Z-оценки; строки, у которых |z\_score| > z\_thresh хотя бы по одному признаку, удаляются.

        После выполнения:

            - self.df обновляется без дубликатов и выбросов.

            - self.features и self.target обновляются согласно очищенному DataFrame.

            - self.scaled\_features заполняется DataFrame с масштабированными признаками.

        Выбрасывает:

            RuntimeError: если self.df не инициализирован.

        """

        if self.df is None:

            raise RuntimeError("Данные не загружены. Сначала вызовите \_load\_data().")

        df\_proc = self.df.copy()

        if drop\_duplicates:

            before\_dupes = df\_proc.shape[0]

            df\_proc = df\_proc.drop\_duplicates().reset\_index(drop=True)

            after\_dupes = df\_proc.shape[0]

            print(f"Удалено дубликатов: {before\_dupes - after\_dupes}")

        if drop\_outliers:

            df\_no\_target = df\_proc.drop(columns=[self.target\_column], errors="ignore")

            means = df\_no\_target.mean()

            stds = df\_no\_target.std(ddof=0)

            z\_scores = (df\_no\_target - means) / stds

            mask = (z\_scores.abs() <= z\_thresh).all(axis=1)

            before\_out = df\_proc.shape[0]

            df\_proc = df\_proc[mask].reset\_index(drop=True)

Продолжение Листинга А

            after\_out = df\_proc.shape[0]

            print(f"Удалено выбросов: {before\_out - after\_out}")

        self.df = df\_proc

        self.target = df\_proc[self.target\_column].copy()

        self.features = df\_proc.drop(columns=[self.target\_column]).copy()

        scaler = StandardScaler()

        scaled\_array = scaler.fit\_transform(self.features)

        self.scaled\_features = pd.DataFrame(

            scaled\_array, columns=self.features.columns, index=self.features.index

        )

        print(

            "Предобработка завершена: дубликаты и выбросы удалены (если указано), признаки масштабированы."

        )

    def visualize\_distributions(self, figsize: Tuple[int, int] = (12, 8)) -> None:

        """

        Строит гистограммы распределений каждого признака (до масштабирования).

        Параметры:

            figsize (Tuple[int, int]): Размер фигуры (ширина, высота) в дюймах.

        """

        if self.features is None:

            raise RuntimeError(

                "Признаки не выделены. Сначала вызовите \_extract\_features\_target()."

            )

        n = len(self.features.columns)

        cols = 3

        rows = (n + cols - 1) // cols

        fig, axes = plt.subplots(rows, cols, figsize=figsize)

        axes = axes.flatten()

        for i, col in enumerate(self.features.columns):

            axes[i].hist(self.features[col], bins=15, edgecolor="black")

            axes[i].set\_title(col)

        for j in range(n, len(axes)):

            axes[j].axis("off")

        plt.tight\_layout()

        plt.show()

    def visualize\_correlation\_heatmap(

        self, figsize: Tuple[int, int] = (18, 12)

    ) -> None:

        """

        Строит тепловую карту корреляций между признаками с помощью seaborn.

        Параметры:

            figsize (Tuple[int, int]): Размер фигуры (ширина, высота) в дюймах.

        Выбрасывает:

            RuntimeError: если self.features не инициализирован.

        """

        if self.features is None:

            raise RuntimeError(

Продолжение Листинга А

                "Признаки не выделены. Вызовите \_extract\_features\_target()."

            )

        plt.figure(figsize=figsize)

        sns.heatmap(

            self.features.corr(),

            annot=True,

            cmap="coolwarm",

            linewidths=0.5,

            square=True,

            cbar\_kws={"shrink": 0.7},

        )

        plt.title("Матрица корреляции признаков")

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

        plt.yticks(rotation=0)

        plt.tight\_layout()

        plt.show()

    def split\_data(self, test\_size: float = 0.2, random\_state: int = 42) -> None:

        """

        Разделяет данные на обучающую и тестовую выборки.

        Параметры:

            test\_size (float): Доля тестовой выборки (по умолчанию 0.2).

            random\_state (int): Seed для воспроизводимости (по умолчанию 42).

        Выбрасывает:

            RuntimeError: если self.scaled\_features или self.target не инициализированы.

        """

        if self.scaled\_features is None or self.target is None:

            raise RuntimeError("Данные не предобработаны. Вызовите preprocess().")

        self.X\_train, self.X\_test, self.y\_train, self.y\_test = train\_test\_split(

            self.scaled\_features,

            self.target,

            test\_size=test\_size,

            random\_state=random\_state,

        )

        print(

            f"Данные разделены:\n"

            f"- Обучающая выборка: {self.X\_train.shape[0]} образцов\n"

            f"- Тестовая выборка: {self.X\_test.shape[0]} образцов"

        )

    def get\_training\_data(self) -> Tuple[DataFrame, Series]:

        """

        Возвращает обучающие данные (признаки и целевую переменную).

        Возвращает:

            Tuple[DataFrame, Series]: (X\_train, y\_train).

        Выбрасывает:

            RuntimeError: если split\_data не был вызван.

        """

        if not hasattr(self, "X\_train") or not hasattr(self, "y\_train"):

            raise RuntimeError("Данные не разделены. Вызовите split\_data().")

        return self.X\_train, self.y\_train

    def get\_testing\_data(self) -> Tuple[DataFrame, Series]:

Продолжение Листинга А

        """

        Возвращает тестовые данные (признаки и целевую переменную).

        Возвращает:

            Tuple[DataFrame, Series]: (X\_test, y\_test).

        Выбрасывает:

            RuntimeError: если split\_data не был вызван.

        """

        if not hasattr(self, "X\_test") or not hasattr(self, "y\_test"):

            raise RuntimeError("Данные не разделены. Вызовите split\_data().")

        return self.X\_test, self.y\_test

    def remove\_feature(self, feature\_name: str) -> None:

        """

        Удаляет указанный признак из текущего набора данных.

        Параметры:

            feature\_name (str): Название удаляемого признака.

        Исключения:

            ValueError: если feature\_name не строка.

            KeyError: если признак отсутствует в self.features.

            RuntimeError: если self.features не инициализирован.

        """

        if self.features is None:

            raise RuntimeError(

                "Набор признаков пуст. Вызовите \_extract\_features\_target()."

            )

        if not isinstance(feature\_name, str):

            raise ValueError(

                f"Имя признака должно быть строкой, получено {type(feature\_name).\_\_name\_\_}"

            )

        if feature\_name not in self.features.columns:

            raise KeyError(

                f"Признак '{feature\_name}' отсутствует в текущем наборе признаков."

            )

        self.features.drop(columns=[feature\_name], inplace=True)

        if self.df is not None and feature\_name in self.df.columns:

            self.df.drop(columns=[feature\_name], inplace=True)

        if (

            self.scaled\_features is not None

            and feature\_name in self.scaled\_features.columns

        ):

            self.scaled\_features.drop(columns=[feature\_name], inplace=True)

        print(f"Признак '{feature\_name}' успешно удалён из набора данных.")

    def visualize\_target\_distribution(

        self,

        figsize: Tuple[int, int] = (10, 6),

        bins: int = 30,

        kde: bool = True,

        log\_scale: bool = False

    ) -> None:

        """

        Строит гистограмму распределения целевой переменной (цены дома).

        Параметры:

            figsize (Tuple[int, int]): Размер графика (ширина, высота).

Продолжение Листинга А

            bins (int): Количество бинов для гистограммы.

            kde (bool): Отображать ли кривую оценки плотности.

            log\_scale (bool): Использовать логарифмическую шкалу для оси Y.

        Выбрасывает:

            RuntimeError: Если целевая переменная не загружена.

        """

        if self.target is None:

            raise RuntimeError("Целевая переменная не загружена. Сначала вызовите \_extract\_features\_target().")

        plt.figure(figsize=figsize)

        sns.histplot(

            self.target,

            bins=bins,

            kde=kde,

            color='skyblue',

            edgecolor='black',

            linewidth=1.2,

            alpha=0.7

        )

        mean\_val = self.target.mean()

        median\_val = self.target.median()

        std\_val = self.target.std()

        plt.axvline(mean\_val, color='red', linestyle='--', linewidth=2, label=f'Среднее: {mean\_val:.2f}')

        plt.axvline(median\_val, color='green', linestyle='-', linewidth=2, label=f'Медиана: {median\_val:.2f}')

        plt.axvline(mean\_val - std\_val, color='purple', linestyle=':', linewidth=1.5, label=f'±1 STD')

        plt.axvline(mean\_val + std\_val, color='purple', linestyle=':', linewidth=1.5)

        if log\_scale:

            plt.yscale('log')

            plt.ylabel('Частота (log scale)')

        else:

            plt.ylabel('Частота')

        plt.title(f'Распределение целевой переменной: {self.target\_column}')

        plt.xlabel(self.target\_column)

        plt.grid(True, alpha=0.3)

        plt.legend()

        stats\_text = (

            f"Минимум: {self.target.min():.2f}\n"

            f"Максимум: {self.target.max():.2f}\n"

            f"Среднее: {mean\_val:.2f}\n"

            f"Медиана: {median\_val:.2f}\n"

            f"Станд. отклонение: {std\_val:.2f}\n"

        )

        plt.annotate(

            stats\_text,

            xy=(0.98, 0.7),

            xycoords='axes fraction',

            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.8),

            horizontalalignment='right',

Окончание Листинга А

            fontsize=10

        )

        plt.tight\_layout()

        plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    # manager = DatasetManager(csv\_path="Student\_Performance.csv", target\_column="Performance Index")

    # stats = manager.compute\_basic\_statistics()

    # print("Описание признаков:")

    # print(stats["describe"])

    # print("\nКорреляционная матрица признаков:")

    # print(stats["correlation\_matrix"])

    # print("\nСтатистика по целевой переменной:")

    # print(stats["target\_distribution"])

    # manager.visualize\_distributions()

    # manager.visualize\_correlation\_heatmap()

    # manager.visualize\_target\_distribution()

    # manager.preprocess(drop\_duplicates=True, drop\_outliers=True, z\_thresh=2.0)

    # manager.split\_data(test\_size=0.2, random\_state=42)

    # X\_train, y\_train = manager.get\_training\_data()

    # X\_test, y\_test = manager.get\_testing\_data()

    test\_manager = DatasetManager('housing\_dataset.csv', 'Price')

    test\_manager.preprocess(z\_thresh=5)

### Приложение Б

Файл linear\_regression.py с использованием готовой реализации алгоритма наименьших квадратов

Листинг Б – Содержание файла dataset\_manager.py

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from typing import Optional, List, Tuple

from pandas import DataFrame, Series

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import (

    mean\_squared\_error,

    mean\_absolute\_error,

    r2\_score,

    median\_absolute\_error,

    explained\_variance\_score,

)

from dataset\_manager import DatasetManager

class RegressionSklearn:

    """

    Класс для линейной регрессии (OLS) с использованием sklearn.

    Атрибуты:

        model (Optional[LinearRegression]): экземпляр обученной модели LinearRegression.

        feature\_names (List[str]): список имён признаков, использованных при обучении.

        fitted (bool): флаг, указывающий, что модель обучена.

    Методы:

        fit(X\_train, y\_train) -> None:

            Обучает модель LinearRegression на тренировочных данных.

        evaluate(X\_test, y\_test) -> float:

            Вычисляет RMSE на тестовой выборке.

        plot\_predicted\_vs\_actual(X, y, figsize=(8,6)) -> None:

            Строит scatter-диаграмму: предсказанные значения vs фактические.

    """

    def \_\_init\_\_(self) -> None:

        self.model: Optional[LinearRegression] = None

        self.feature\_names: List[str] = []

        self.fitted: bool = False

    def fit(self, X\_train: DataFrame, y\_train: Series) -> None:

        """

        Обучает линейную регрессию (LinearRegression) на тренировочных данных.

        Параметры:

            X\_train (DataFrame): DataFrame с признаками для обучения.

            y\_train (Series): Series с целевой переменной для обучения.

        Исключения:

            ValueError: если X\_train не DataFrame или y\_train не Series,

                        либо их длины не совпадают.

        """

        if not isinstance(X\_train, pd.DataFrame):

            raise ValueError("X\_train должен быть pandas.DataFrame")

Продолжение Листинга Б

        if not isinstance(y\_train, pd.Series):

            raise ValueError("y\_train должен быть pandas.Series")

        if len(X\_train) != len(y\_train):

            raise ValueError("X\_train и y\_train должны быть одной длины")

        self.feature\_names = list(X\_train.columns)

        self.model = LinearRegression()

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

        self.fitted = True

    def evaluate(self, X\_test: DataFrame, y\_test: Series) -> float:

        """

        Вычисляет RMSE (root mean squared error) на тестовой выборке.

        Параметры:

            X\_test (DataFrame): DataFrame с признаками тестовой выборки.

            y\_test (Series): Series с целевой переменной тестовой выборки.

        Возвращает:

            float: Значение RMSE на тестовых данных.

        Исключения:

            RuntimeError: если модель не обучена.

            ValueError: если X\_test/y\_test не тех типов или длины не совпадают.

        """

        if not self.fitted or self.model is None:

            raise RuntimeError("Модель не обучена. Сначала вызовите fit().")

        if not isinstance(X\_test, pd.DataFrame):

            raise ValueError("X\_test должен быть pandas.DataFrame")

        if not isinstance(y\_test, pd.Series):

            raise ValueError("y\_test должен быть pandas.Series")

        if len(X\_test) != len(y\_test):

            raise ValueError("X\_test и y\_test должны быть одной длины")

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

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

        rmse = np.sqrt(mean\_squared\_error(y\_test.values, y\_pred))

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

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

        evs = explained\_variance\_score(y\_test, y\_pred)

        mdae = median\_absolute\_error(y\_test, y\_pred)

        return {

            "mse": mse,

            "rmse": rmse,

            "mae": mae,

            "mdae": mdae,

            "r2": r2,

            "evs": evs,

        }

    def plot\_predicted\_vs\_actual(

        self, X: DataFrame, y: Series, figsize: Tuple[int, int] = (8, 6)

    ) -> None:

        """

        Строит scatter-диаграмму: предсказанные значения vs фактические.

        Параметры:

            X (DataFrame): DataFrame признаков для предсказания.

Продолжение Листинга Б

            y (Series): Series фактических значений целевой переменной.

            figsize (Tuple[int, int]): Размер фигуры (ширина, высота) в дюймах.

        Исключения:

            RuntimeError: если модель не обучена.

            ValueError: если X/y не тех типов или длины не совпадают.

        """

        if not self.fitted or self.model is None:

            raise RuntimeError("Модель не обучена. Сначала вызовите fit().")

        if not isinstance(X, pd.DataFrame):

            raise ValueError("X должен быть pandas.DataFrame")

        if not isinstance(y, pd.Series):

            raise ValueError("y должен быть pandas.Series")

        if len(X) != len(y):

            raise ValueError("X и y должны быть одной длины")

        preds = self.model.predict(X)

        plt.figure(figsize=figsize)

        plt.scatter(y.values, preds, alpha=0.6, edgecolor="k")

        plt.plot([y.min(), y.max()], [y.min(), y.max()], color="red", linewidth=2)

        plt.title("Предсказанные значения vs фактические")

        plt.xlabel("Фактические значения")

        plt.ylabel("Предсказанные значения")

        plt.tight\_layout()

        plt.show()

    def plot\_feature\_importance(

        self,

        figsize: Tuple[int, int] = (12, 8),

        importance\_type: str = "coefficient",

        color: str = "skyblue",

    ) -> None:

        """

        Визуализирует важность признаков на основе коэффициентов линейной регрессии.

        Параметры:

            figsize (Tuple[int, int]): Размер графика (ширина, высота).

            importance\_type (str): Тип важности:

                'coefficient' - исходные коэффициенты

                'absolute' - абсолютные значения коэффициентов

                'normalized' - нормализованные абсолютные значения (сумма=100%)

            color (str): Цвет столбцов диаграммы.

        """

        features = self.model.feature\_names\_in\_

        coefs = self.model.coef\_

        if importance\_type == "coefficient":

            importance = coefs

            ylabel = "Коэффициент"

        elif importance\_type == "absolute":

            importance = np.abs(coefs)

            ylabel = "Абсолютное значение коэффициента"

        elif importance\_type == "normalized":

            importance = 100 \* np.abs(coefs) / np.sum(np.abs(coefs))

            ylabel = "Относительная важность (%)"

        else:

            raise ValueError(

                f"Неизвестный тип важности: {importance\_type}. "

Окончание Листинга Б

                "Допустимые значения: 'coefficient', 'absolute', 'normalized'"

            )

        importance\_df = pd.DataFrame({"feature": features, "importance": importance})

        importance\_df = importance\_df.sort\_values(

            by="importance",

            ascending=True if importance\_type == "coefficient" else False,

        )

        plt.figure(figsize=figsize)

        bars = plt.barh(

            importance\_df["feature"],

            importance\_df["importance"],

            color=color,

            edgecolor="black",

        )

        for bar in bars:

            width = bar.get\_width()

            plt.annotate(

                f"{width:.4f}",

                xy=(width, bar.get\_y() + bar.get\_height() / 2),

                xytext=(3, 0),

                textcoords="offset points",

                ha="left",

                va="center",

                fontsize=10,

            )

        plt.xlabel(ylabel)

        plt.title("Важность признаков в линейной регрессии")

        plt.grid(axis="x", alpha=0.3)

        plt.tight\_layout()

        plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    manager = DatasetManager(

        csv\_path="Student\_Performance.csv", target\_column="Performance Index"

    )

    manager.preprocess()

    manager.split\_data()

    X\_train, y\_train = manager.get\_training\_data()

    X\_test, y\_test = manager.get\_testing\_data()

    reg = RegressionSklearn()

    reg.fit(X\_train, y\_train)

    metrics = reg.evaluate(X\_test, y\_test)

    reg.plot\_predicted\_vs\_actual(X\_test, y\_test)

    reg.plot\_feature\_importance(importance\_type="coefficient")

    print("Результаты оценки модели:")

    for metric, value in metrics.items():

        print(f"{metric.upper()}: {value:.2f}")

### Приложение В

Файл linear\_regression\_custom.py с собственной реализацией стохастического градиентного спуска

Листинг В – Файл linear\_regression\_custom.py

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from typing import Tuple, Optional, Dict, Union

from dataset\_manager import DatasetManager

class CustomLinearRegression:

    """

    Кастомная реализация многомерной линейной регрессии с использованием стохастического градиентного спуска.

    Параметры:

        learning\_rate (float): Скорость обучения (по умолчанию 0.01)

        epochs (int): Количество эпох обучения (по умолчанию 1000)

        batch\_size (int): Размер батча для мини-пакетного обучения (по умолчанию 32)

        random\_state (int): Seed для воспроизводимости (по умолчанию None)

        early\_stopping (bool): Останавливать ли обучение при ухудшении ошибки (по умолчанию True)

        patience (int): Количество эпох для ранней остановки (по умолчанию 10)

    Атрибуты:

        weights (np.ndarray): Веса модели (включая смещение)

        feature\_names (list): Имена признаков

        errors (list): История ошибок в процессе обучения

        best\_weights (np.ndarray): Лучшие веса (при использовании ранней остановки)

    """

    def \_\_init\_\_(

        self,

        learning\_rate: float = 0.01,

        epochs: int = 1000,

        batch\_size: int = 32,

        random\_state: Optional[int] = None,

        early\_stopping: bool = True,

        patience: int = 10,

    ) -> None:

        self.learning\_rate = learning\_rate

        self.epochs = epochs

        self.batch\_size = batch\_size

        self.random\_state = random\_state

        self.early\_stopping = early\_stopping

        self.patience = patience

        if random\_state is not None:

            np.random.seed(random\_state)

        self.weights = None

        self.feature\_names = None

        self.errors = []

        self.best\_weights = None

        self.best\_error = float("inf")

Продолжение Листинга В

    def \_initialize\_weights(self, n\_features: int) -> None:

        """Инициализирует веса случайными малыми значениями"""

        self.weights = np.random.randn(n\_features + 1) \* 0.01

    def \_add\_bias(self, X: np.ndarray) -> np.ndarray:

        """Добавляет столбец единиц для смещения (bias)"""

        return np.c\_[np.ones(X.shape[0]), X]

    def \_calculate\_loss(self, X: np.ndarray, y: np.ndarray) -> float:

        """Вычисляет среднеквадратичную ошибку (MSE)"""

        predictions = X.dot(self.weights)

        return np.mean((predictions - y) \*\* 2)

    def \_gradient\_step(self, X\_batch: np.ndarray, y\_batch: np.ndarray) -> None:

        """Выполняет один шаг градиентного спуска"""

        predictions = X\_batch.dot(self.weights)

        error = predictions - y\_batch

        gradient = X\_batch.T.dot(error) / len(X\_batch)

        self.weights -= self.learning\_rate \* gradient

    def fit(

        self, X: Union[np.ndarray, pd.DataFrame], y: Union[np.ndarray, pd.Series]

    ) -> None:

        """

        Обучает модель линейной регрессии на предоставленных данных.

        Параметры:

            X (Union[np.ndarray, pd.DataFrame]): Матрица признаков

            y (Union[np.ndarray, pd.Series]): Вектор целевых значений

        Исключения:

            ValueError: Если размеры X и y не совпадают

        """

        if len(X) != len(y):

            raise ValueError("Размеры X и y должны совпадать")

        if isinstance(X, pd.DataFrame):

            self.feature\_names = X.columns.tolist()

        else:

            self.feature\_names = [f"feature\_{i}" for i in range(X.shape[1])]

        X = np.array(X)

        y = np.array(y)

        self.\_initialize\_weights(X.shape[1])

        X\_with\_bias = self.\_add\_bias(X)

        no\_improvement\_count = 0

        self.errors = []

        self.best\_weights = self.weights.copy()

        self.best\_error = float("inf")

        for \_ in range(self.epochs):

            indices = np.arange(len(X))

            np.random.shuffle(indices)

            X\_shuffled = X\_with\_bias[indices]

            y\_shuffled = y[indices]

            for i in range(0, len(X), self.batch\_size):

                end = min(i + self.batch\_size, len(X))

Продолжение Листинга В

                X\_batch = X\_shuffled[i:end]

                y\_batch = y\_shuffled[i:end]

                self.\_gradient\_step(X\_batch, y\_batch)

            current\_error = self.\_calculate\_loss(X\_with\_bias, y)

            self.errors.append(current\_error)

            if self.early\_stopping:

                if current\_error < self.best\_error:

                    self.best\_error = current\_error

                    self.best\_weights = self.weights.copy()

                    no\_improvement\_count = 0

                else:

                    no\_improvement\_count += 1

                if no\_improvement\_count >= self.patience:

                    self.weights = self.best\_weights

                    break

        if not self.early\_stopping:

            self.best\_weights = self.weights.copy()

    def predict(self, X: Union[np.ndarray, pd.DataFrame]) -> np.ndarray:

        """

        Предсказывает значения для новых данных.

        Параметры:

            X (Union[np.ndarray, pd.DataFrame]): Матрица признаков для предсказания

        Возвращает:

            np.ndarray: Предсказанные значения

        Исключения:

            RuntimeError: Если модель не обучена

        """

        if self.weights is None:

            raise RuntimeError("Модель не обучена. Сначала вызовите fit()")

        X = np.array(X)

        X\_with\_bias = self.\_add\_bias(X)

        return X\_with\_bias.dot(self.weights)

    def evaluate(

        self, X: Union[np.ndarray, pd.DataFrame], y: Union[np.ndarray, pd.Series]

    ) -> Dict[str, float]:

        """

        Оценивает модель на тестовых данных и возвращает комплекс метрик регрессии.

        Параметры:

            X (Union[np.ndarray, pd.DataFrame]): Матрица признаков

            y (Union[np.ndarray, pd.Series]): Вектор целевых значений

        Возвращает:

            Dict[str, float]: Словарь с метриками:

                - 'mse': Средняя квадратичная ошибка

                - 'rmse': Корень из средней квадратичной ошибки

                - 'mae': Средняя абсолютная ошибка

Продолжение Листинга В

                - 'r2': Коэффициент детерминации (R²)

                - 'explained\_variance': Объясненная дисперсия

        """

        y\_pred = self.predict(X)

        y\_true = np.array(y)

        mse = np.mean((y\_true - y\_pred) \*\* 2)

        rmse = np.sqrt(mse)

        mae = np.mean(np.abs(y\_true - y\_pred))

        ss\_total = np.sum((y\_true - np.mean(y\_true)) \*\* 2)

        ss\_res = np.sum((y\_true - y\_pred) \*\* 2)

        r2 = 1 - (ss\_res / ss\_total)

        explained\_variance = 1 - (np.var(y\_true - y\_pred) / np.var(y\_true))

        return {

            "mse": mse,

            "rmse": rmse,

            "mae": mae,

            "r2": r2,

            "explained\_variance": explained\_variance,

        }

    def get\_feature\_importance(self) -> Dict[str, float]:

        """

        Возвращает важность признаков на основе абсолютных значений весов.

        Возвращает:

            Dict[str, float]: Словарь {имя\_признака: важность}

        Исключения:

            RuntimeError: Если модель не обучена

        """

        if self.weights is None:

            raise RuntimeError("Модель не обучена. Сначала вызовите fit()")

        feature\_weights = self.weights[1:]

        abs\_weights = np.abs(feature\_weights)

        normalized\_importance = 100 \* abs\_weights / np.sum(abs\_weights)

        return dict(zip(self.feature\_names, normalized\_importance))

    def plot\_learning\_curve(self, figsize: Tuple[int, int] = (10, 6)) -> None:

        """

        Визуализирует кривую обучения (ошибка vs эпохи).

        Исключения:

            RuntimeError: Если модель не обучена

        """

        if not self.errors:

            raise RuntimeError("История ошибок недоступна. Сначала обучите модель")

        plt.figure(figsize=figsize)

        plt.plot(self.errors)

        plt.title("Кривая обучения")

        plt.xlabel("Эпоха")

        plt.ylabel("MSE")

        plt.grid(True, alpha=0.3)

Продолжение Листинга В

        plt.show()

    def plot\_predictions\_vs\_actual(

        self,

        X: Union[np.ndarray, pd.DataFrame],

        y: Union[np.ndarray, pd.Series],

        figsize: Tuple[int, int] = (8, 6),

    ) -> None:

        """

        Визуализирует предсказанные значения vs фактические значения.

        Исключения:

            RuntimeError: Если модель не обучена

        """

        y\_pred = self.predict(X)

        y\_true = np.array(y)

        plt.figure(figsize=figsize)

        plt.scatter(y\_true, y\_pred, alpha=0.6, edgecolor="k")

        min\_val = min(y\_true.min(), y\_pred.min())

        max\_val = max(y\_true.max(), y\_pred.max())

        plt.plot([min\_val, max\_val], [min\_val, max\_val], color="red", linewidth=2)

        plt.title("Предсказанные vs Фактические значения")

        plt.xlabel("Фактические значения")

        plt.ylabel("Предсказанные значения")

        plt.grid(True, alpha=0.3)

        plt.show()

    def plot\_feature\_importance(

        self, figsize: Tuple[int, int] = (10, 6), top\_n: Optional[int] = None

    ) -> None:

        """

        Визуализирует важность признаков на основе абсолютных значений весов.

        Параметры:

            top\_n (int): Количество топ-признаков для отображения (по умолчанию все)

        Исключения:

            RuntimeError: Если модель не обучена

        """

        importance = self.get\_feature\_importance()

        sorted\_importance = sorted(importance.items(), key=lambda x: x[1], reverse=True)

        if top\_n is not None:

            sorted\_importance = sorted\_importance[:top\_n]

        features, importances = zip(\*sorted\_importance)

        plt.figure(figsize=figsize)

        plt.barh(features, importances, color="skyblue")

        plt.title("Важность признаков")

        plt.xlabel("Важность (%)")

        plt.grid(axis="x", alpha=0.3)

        plt.tight\_layout()

        plt.show()

Окончание Листинга В

    def get\_coefficients(self) -> Dict[str, float]:

        """

        Возвращает коэффициенты модели для каждого признака.

        Возвращает:

            Dict[str, float]: Словарь {имя\_признака: коэффициент}

        Исключения:

            RuntimeError: Если модель не обучена

        """

        if self.weights is None:

            raise RuntimeError("Модель не обучена. Сначала вызовите fit()")

        feature\_weights = self.weights[1:]

        return dict(zip(self.feature\_names, feature\_weights))

manager = DatasetManager(

    csv\_path="Student\_Performance.csv", target\_column="Performance Index"

)

manager.preprocess()

manager.split\_data()

X\_train, y\_train = manager.get\_training\_data()

X\_test, y\_test = manager.get\_testing\_data()

model = CustomLinearRegression(

    learning\_rate=0.01,

    epochs=1000,

    batch\_size=32,

    random\_state=42,

    early\_stopping=True,

    patience=20,

)

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

test\_metrics = model.evaluate(X\_test, y\_test)

print("\nМетрики на тестовой выборке:")

for metric, value in test\_metrics.items():

    print(f"{metric}: {value:.4f}")

model.plot\_learning\_curve()

model.plot\_predictions\_vs\_actual(X\_test, y\_test)

model.plot\_feature\_importance(top\_n=10)

coefficients = model.get\_coefficients()

print("\nКоэффициенты модели:")

for feature, coef in coefficients.items():

    print(f"{feature}: {coef:.4f}")

### Приложение Г

Файл Lasso\_regression.py с использованием готовой реализацией регрессии Лассо

Листинг Г – Файл Lasso\_regression.py

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from typing import Optional, List, Tuple, Dict

from pandas import DataFrame, Series

from sklearn.linear\_model import Lasso

from sklearn.metrics import (

    mean\_squared\_error,

    mean\_absolute\_error,

    r2\_score,

    median\_absolute\_error,

    explained\_variance\_score,

)

from dataset\_manager import DatasetManager

class LassoRegressionSklearn:

    def \_\_init\_\_(self) -> None:

        """

        Инициализирует пустые атрибуты модели.

        """

        self.model: Optional[Lasso] = None

        self.feature\_names: List[str] = []

        self.fitted: bool = False

    def fit(

        self,

        X\_train: DataFrame,

        y\_train: Series,

        alpha: float = 1.0,

        max\_iter: int = 1000,

        random\_state: Optional[int] = None,

    ) -> None:

        """

        Обучает модель Lasso (sklearn.linear\_model.Lasso) на тренировочных данных.

        Параметры:

            X\_train (DataFrame): DataFrame с признаками для обучения.

            y\_train (Series): Series с целевой переменной для обучения.

            alpha (float): Коэффициент регуляризации L1 (по умолчанию 1.0).

            max\_iter (int): Максимальное число итераций для оптимизации (по умолчанию 1000).

            random\_state (Optional[int]): Сид для воспроизводимости (по умолчанию None).

        Исключения:

            ValueError: если X\_train не pandas.DataFrame или y\_train не pandas.Series,

                        либо их длины не совпадают.

        """

        if not isinstance(X\_train, pd.DataFrame):

            raise ValueError("X\_train должен быть pandas.DataFrame")

        if not isinstance(y\_train, pd.Series):

            raise ValueError("y\_train должен быть pandas.Series")

Продолжение Листинга Г

        if len(X\_train) != len(y\_train):

            raise ValueError("X\_train и y\_train должны быть одной длины")

        self.feature\_names = list(X\_train.columns)

        self.model = Lasso(alpha=alpha, max\_iter=max\_iter, random\_state=random\_state)

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

        self.fitted = True

    def evaluate(self, X\_test: DataFrame, y\_test: Series) -> Dict[str, float]:

        """

        Вычисляет основные метрики качества на тестовой выборке:

            - MSE (mean squared error)

            - RMSE (root mean squared error)

            - MAE (mean absolute error)

            - MdAE (median absolute error)

            - R2 score (коэффициент детерминации)

            - EVS (explained variance score)

        Параметры:

            X\_test (DataFrame): DataFrame с признаками тестовой выборки.

            y\_test (Series): Series с целевой переменной тестовой выборки.

        Возвращает:

            Dict[str, float]: Словарь с ключами ["mse", "rmse", "mae", "mdae", "r2", "evs"].

        Исключения:

            RuntimeError: если модель не обучена.

            ValueError: если X\_test/y\_test не тех типов или длины не совпадают.

        """

        if not self.fitted or self.model is None:

            raise RuntimeError("Модель не обучена. Сначала вызовите fit().")

        if not isinstance(X\_test, pd.DataFrame):

            raise ValueError("X\_test должен быть pandas.DataFrame")

        if not isinstance(y\_test, pd.Series):

            raise ValueError("y\_test должен быть pandas.Series")

        if len(X\_test) != len(y\_test):

            raise ValueError("X\_test и y\_test должны быть одной длины")

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

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

        rmse = np.sqrt(mse)

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

        mdae = median\_absolute\_error(y\_test, y\_pred)

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

        evs = explained\_variance\_score(y\_test, y\_pred)

        return {

            "mse": mse,

            "rmse": rmse,

            "mae": mae,

            "mdae": mdae,

            "r2": r2,

            "evs": evs,

        }

    def plot\_predicted\_vs\_actual(

        self, X: DataFrame, y: Series, figsize: Tuple[int, int] = (8, 6)

    ) -> None:

Продолжение Листинга Г

        """

        Строит scatter-диаграмму: предсказанные значения vs фактические.

        Параметры:

            X (DataFrame): DataFrame с признаками для предсказания.

            y (Series): Series с фактическими значениями целевой переменной.

            figsize (Tuple[int, int]): Размер фигуры (ширина, высота) в дюймах.

        Исключения:

            RuntimeError: если модель не обучена.

            ValueError: если X/y не тех типов или длины не совпадают.

        """

        if not self.fitted or self.model is None:

            raise RuntimeError("Модель не обучена. Сначала вызовите fit().")

        if not isinstance(X, pd.DataFrame):

            raise ValueError("X должен быть pandas.DataFrame")

        if not isinstance(y, pd.Series):

            raise ValueError("y должен быть pandas.Series")

        if len(X) != len(y):

            raise ValueError("X и y должны быть одной длины")

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

        plt.figure(figsize=figsize)

        plt.scatter(y.values, y\_pred, alpha=0.6, edgecolor="k")

        plt.plot([y.min(), y.max()], [y.min(), y.max()], color="red", linewidth=2)

        plt.title("Предсказанные значения vs фактические (Lasso)")

        plt.xlabel("Фактические значения")

        plt.ylabel("Предсказанные значения")

        plt.tight\_layout()

        plt.show()

    def plot\_feature\_importance(

        self,

        figsize: Tuple[int, int] = (12, 8),

        importance\_type: str = "coefficient",

        color: str = "skyblue",

    ) -> None:

        """

        Визуализирует важность признаков на основе коэффициентов Lasso-модели.

        Параметры:

            figsize (Tuple[int, int]): Размер графика (ширина, высота).

            importance\_type (str): Тип важности:

                - 'coefficient'  : исходные коэффициенты (с учётом знака)

                - 'absolute'     : абсолютные значения коэффициентов

                - 'normalized'   : относительные абсолютные (в сумме = 100%)

            color (str): Цвет столбцов диаграммы.

        Исключения:

            RuntimeError: если модель не обучена.

            ValueError: если importance\_type задан неверно.

        """

        if not self.fitted or self.model is None:

            raise RuntimeError("Модель не обучена. Сначала вызовите fit().")

        coefs = self.model.coef\_

        features = np.array(self.feature\_names)

        if importance\_type == "coefficient":

            importance = coefs

Продолжение Листинга Г

            ylabel = "Коэффициент"

        elif importance\_type == "absolute":

            importance = np.abs(coefs)

            ylabel = "Абсолютное значение коэффициента"

        elif importance\_type == "normalized":

            abs\_vals = np.abs(coefs)

            if abs\_vals.sum() == 0:

                importance = abs\_vals

            else:

                importance = 100 \* abs\_vals / abs\_vals.sum()

            ylabel = "Относительная важность (%)"

        else:

            raise ValueError(

                f"Неизвестный тип важности: {importance\_type}. "

                "Допустимые: 'coefficient', 'absolute', 'normalized'"

            )

        importance\_df = pd.DataFrame({"feature": features, "importance": importance})

        ascending = True if importance\_type == "coefficient" else False

        importance\_df = importance\_df.sort\_values(by="importance", ascending=ascending)

        plt.figure(figsize=figsize)

        bars = plt.barh(

            importance\_df["feature"],

            importance\_df["importance"],

            color=color,

            edgecolor="black",

        )

        for bar in bars:

            width = bar.get\_width()

            plt.annotate(

                f"{width:.4f}",

                xy=(width, bar.get\_y() + bar.get\_height() / 2),

                xytext=(3, 0),

                textcoords="offset points",

                ha="left",

                va="center",

                fontsize=10,

            )

        plt.xlabel(ylabel)

        plt.title("Важность признаков в Lasso-регрессии")

        plt.grid(axis="x", alpha=0.3)

        plt.tight\_layout()

        plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    manager = DatasetManager(

        csv\_path="Student\_Performance.csv", target\_column="Performance Index"

    )

    manager.preprocess()

    manager.split\_data(test\_size=0.2, random\_state=42)

    X\_train, y\_train = manager.get\_training\_data()

    X\_test, y\_test = manager.get\_testing\_data()

    lasso\_reg = LassoRegressionSklearn()

Окончание Листинга Г

    lasso\_reg.fit(X\_train, y\_train, alpha=0.1, max\_iter=1000, random\_state=42)

    metrics = lasso\_reg.evaluate(X\_test, y\_test)

    print("Результаты оценки Lasso-модели:")

    for name, value in metrics.items():

        print(f"{name.upper()}: {value:.4f}")

    lasso\_reg.plot\_predicted\_vs\_actual(X\_test, y\_test)

    lasso\_reg.plot\_feature\_importance(importance\_type="absolute", color="teal")