Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий – РТФ

Школа бакалавриата

ОТЧЕТ

По проекту
«Разработка образовательных материалов и проектов в сфере Data Science»

по дисциплине «Проектный практикум»

Заказчик: Ильинский А.Д.

Куратор: Ильинский А.Д.

Студенты команды

Паюсов М.Ю.

Чижевская М.И.

Екатеринбург, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В рамках данного проекта будут реализованы два учебных исследования: анализ данных о выживших пассажирах «Титаника» и анализ популярности треков на стриминговой платформе Spotify. Оба исследования направлены на освоение базовых принципов машинного обучения и практических навыков анализа данных.

Цель проекта: освоить методы предварительной обработки, анализа и визуализации данных, а также научиться применять модели машинного обучения к задачам классификации и регрессии на реальных наборах данных.

Задачи проекта:

1) изучить структуру предоставленных датасетов и провести их предварительную очистку;
2) выполнить визуальный анализ данных и выделить значимые признаки;
3) построить и обучить модели машинного обучения;
   а) для задачи классификации (прогноз выживаемости на основе данных Titanic);
   б) для задачи регрессии (прогноз популярности музыкальных треков Spotify).
4) провести оценку качества моделей и сделать интерпретируемые выводы на основе результатов.

Актуальность: Обработка и анализ данных являются ключевыми этапами в большинстве современных цифровых систем. Задачи классификации и регрессии лежат в основе рекомендаций, прогнозов, персонализации сервисов. Примеры Titanic и Spotify позволяют на доступных,

но практических кейсах изучить жизненный цикл ML-проекта — от загрузки данных до оценки модели.

Область применения:

Разработанные модели и навыки применимы в таких сферах, как:

– анализ пользовательского поведения и прогнозирование предпочтений (на примере Spotify);

– принятие решений в условиях неопределенности (на примере Titanic);

– построение рекомендательных систем и оценка рисков.

Ожидаемые результаты:

– формирование устойчивых навыков обработки и анализа данных;

– умение применять инструменты машинного обучения к типовым задачам;

– получение практического опыта работы с задачами классификации и регрессии;

– повышение уровня цифровой грамотности и компетенций в сфере Data Science.

1 Основная часть

1.1 Работа каждого участника

Проект выполнялся в составе команды из двух участников. Работы над проектом велись совместно, с четким разделением зон ответственности, что позволило эффективно распределить задачи и организовать процесс. На первом этапе оба участника участвовали в подготовке данных и первичном исследовательском анализе (EDA).

Участник 1 (Чижевская Милана Игоревна):

– отвечал за визуализацию данных на этапе EDA;

– разрабатывал графики распределений, гистограммы, тепловые карты и диаграммы взаимосвязей между признаками;

– проводил сравнение категориальных признаков с целевой переменной;

– оформлял результаты анализа в понятной и наглядной форме;

– занимался финальным оформлением Jupyter-ноутбуков: структура кода, заголовки, поясняющие комментарии, читаемость вывода, оформление отчета.

Участник 2 (Паюсов Максим Юрьевич):

– фокусировался на построении моделей машинного обучения;

– проводил подготовку данных для моделей: выбор признаков, нормализация, кодирование категориальных данных;

– реализовывал модели классификации (для Titanic) и регрессии (для Spotify), сравнивал различные алгоритмы, настраивал параметры;

– выполнял анализ метрик качества, интерпретацию результатов и оценку важности признаков;

– писал выводы по результатам моделей, делал сравнительный анализ подходов.

Совместная работа была построена через регулярное обсуждение промежуточных результатов, распределение задач, корректировку действий и взаимную помощь. Такой формат способствовал более глубокому пониманию процессов и достижению качественного результата.

1.2 Разбор требований заказчика и пользователей

1.2.1 Разбор требований заказчика

Заказчиком проекта выступает куратор образовательной программы по направлению Data Science.

Основные требования к создаваемому программному продукту были следующими:

1) демонстрация популярных приёмов машинного обучения (ML) ;

Ожидается, что в рамках проекта будут продемонстрированы ключевые этапы работы с данными и моделей машинного обучения.

Это включает:

1.1) предобработку данных — очистка данных, удаление или обработка пропущенных значений, нормализация, масштабирование признаков, кодирование категориальных переменных;

1.2) исследовательский анализ данных (EDA) — визуализация распределений, построение диаграмм корреляции, выявление аномалий, трендов и взаимосвязей;

1.3) обучение моделей — применение базовых алгоритмов: логистическая регрессия (в задаче классификации) и линейная регрессия (в задаче прогнозирования численного значения);

1.4) оценку качества моделей — использование соответствующих метрик: accuracy, precision, recall, F1 (для классификации); MAE, MSE, R^2 (для регрессии);

1.5) интерпретацию результатов — формулировка выводов на основе предсказаний модели, анализ влияния признаков и обоснование корректности применённого подхода.

2) реализация заданий на актуальные темы;

Выбор тем должен быть основан на их востребованности и популярности в образовательной и профессиональной среде:

2.1) titanic — классическая задача бинарной классификации на исторических данных, часто используется в обучающих курсах как пример полной цепочки ML-проекта;

2.2) spotify — реальный набор музыкальных данных, позволяющий практиковаться в регрессии, предсказывая показатель популярности трека по множеству признаков.

Такая тематика обеспечивает как интерес обучающихся, так и прикладную направленность проекта.

3) работа в среде Jupyter Notebook;

Результат проекта должен быть представлен в формате Jupyter Notebook, поскольку он:

а) позволяет удобно совмещать код, графики и текст;

б) является стандартом в обучении Data Science;

в) обеспечивает интерактивность и последовательное выполнение блоков кода;

г) подходит как для демонстрации, так и для самостоятельной работы студентов.

4) обоснованность и логичность решений;

Каждый этап проекта должен сопровождаться:

а) подробным объяснением действий;

б) визуальными подтверждениями (графики, таблицы, диаграммы);

в) аргументированными выводами;

г) последовательным и логически выстроенным оформлением.

Таким образом, ноутбук должен быть не просто демонстрацией, а полноценным учебным пособием, понятным даже новичку.

5) возможность повторного использования и масштабируемость.

Проект должен быть разработан с учетом возможности:

а) адаптации под другие датасеты и задачи (например, заменой данных в аналогичной структуре);

б) включения в портфолио обучающихся как пример реализованного ML-проекта;

в) использования в образовательных курсах как практического пособия.

1.2.2 Разбор требований пользователей

Пользователями выступают студенты, начинающие специалисты и все, кто осваивает основы машинного обучения (ML). Их требования к создаваемому продукту можно выделить следующим образом:

1) доступность и понятность материала;

Пользователи ожидают, что:

а) все этапы будут объяснены простым и доступным языком;

б) ноутбуки будут содержать пошаговые пояснения, позволяющие следить за логикой построения модели;

в) термины и методы будут объяснены прямо в ноутбуке, без необходимости обращаться к дополнительным источникам;

Для многих пользователей это может быть первый опыт работы с ML, поэтому важно избегать перегрузки формулами и профессиональной терминологией без предварительного объяснения.

2) визуализация и интерактивность;

Большинство начинающих пользователей лучше воспринимают информацию визуально, поэтому:

а) графики, диаграммы и тепловые карты должны быть частью анализа;

б) каждая визуализация должна сопровождаться кратким описанием сути и интерпретацией;

в) наглядные примеры и кейсы повышают интерес и вовлечённость в обучение.

3) возможность повторного воспроизведения;

Пользователям важно, чтобы они могли:

а) самостоятельно воспроизвести код, запуская блоки в Jupyter Notebook без дополнительных настроек;

б) адаптировать ноутбуки под свои данные и задачи;

в) использовать проект как шаблон или образец для собственных проектов.

4) чистота и читаемость кода.

Особое внимание пользователи уделяют:

а) структуре кода — он должен быть логически организован по этапам;

б) названиям переменных — они должны быть осмысленными;

Таким образом, конечные пользователи проекта заинтересованы прежде всего в качественном обучающем материале, который сочетает в себе теорию, практику, визуализацию, обоснование и логическую структуру. Ноутбуки должны не просто демонстрировать применение алгоритмов, а учить мышлению и подходу к решению задач машинного обучения.

1.3 Составление плана действий для достижения цели (backlog)

Для реализации учебного проекта, включающего два задания (Titanic и Spotify), был разработан план действий, включающий последовательность этапов, необходимых для достижения целей и соответствия требованиям заказчика и пользователей. Основная цель — создать два Jupyter-ноутбука, демонстрирующих популярные приёмы работы с машинным обучением и ориентированных на прикладные задачи.

Этапы разработки:

1) выбор и понимание задач;

1.1) определение типа задач: классификация (Titanic) и регрессия (Spotify);

1.2) изучение структур данных и постановка целей по каждой задаче;

1.3) формулировка гипотез, которые будут проверяться в процессе анализа.

2) разделение задач между участниками;

2.1) один участник фокусируется на визуализациях и презентации, другой — на моделях и интерпретации результатов.

3) сбор и подготовка данных;

3.1) загрузка датасетов Titanic и Spotify;

3.2) первичный анализ структуры данных, выявление признаков и целевой переменной;

3.3) предобработка данных: работа с пропущенными значениями, преобразование категориальных признаков, удаление лишних столбцов.

4) исследовательский анализ данных (EDA);

4.1) визуализация распределения признаков;

4.2) поиск зависимостей и корреляций между признаками;

4.3) формирование промежуточных выводов по гипотезам.

5) построение и обучение моделей;

5.1) для Titanic: логистическая регрессия, оценка по accuracy, confusion matrix;

5.2) для Spotify: линейная регрессия, оценка по метрикам R^2 и MAE;

5.3) Выбор параметров моделей и обучение на обучающей выборке;

6) оценка качества и доработка моделей;

6.1) тестирование моделей на отложенных данных;

6.2) оценка метрик, визуализация результатов;

6.3) выводы по эффективности моделей.

7) оформление результатов;

7.1) подробные пояснения к каждому этапу;

7.2) комментарии к коду;

7.3) добавление графиков, таблиц, выводов;

7.4) вывод финальных заключений.

8) контроль качества;

8.1) тестирование воспроизводимости кода;

8.2) исправление ошибок и финальная полировка.

Итог: Backlog позволил разбить проект на логически завершённые этапы, определить зоны ответственности и контролировать прогресс. Это обеспечило эффективное сотрудничество, гибкость при внесении изменений и соответствие всем требованиям проекта — как техническим, так и образовательным.

1.4 Анализ аналогов

При реализации учебных заданий по машинному обучению (Titanic и Spotify) были проанализированы существующие аналоги — как открытые обучающие проекты, так и реальные промышленные решения, применяющие методы анализа данных и прогнозирования. Это позволило выработать подходы к моделированию, выбору признаков и структуре проекта, соответствующие современным стандартам в области Data Science.

1.4.1 Исследование Titanic — задача классификации

Аналог «Kaggle: Titanic - Machine Learning from Disaster»

Это один из самых популярных учебных конкурсов по машинному обучению. Он служит отправной точкой для начинающих специалистов в области Data Science. Участникам предлагается классифицировать пассажиров на выживших и невыживших по данным о возрасте, поле, классе билета и другим признакам.

Мы проанализировали десятки успешных решений на Kaggle, что позволило:

– понять важность отбора признаков (например, класс, пол, возраст);

– определить эффективные методы обработки пропусков и кодирования категориальных переменных;

– выбрать стартовую модель (логистическая регрессия) для реализации.

Вывод:

Наша реализация опирается на принципы и этапы, аналогичные тем, что применяются в решениях с высокими рейтингами. При этом мы адаптировали подход под учебную задачу, делая акцент на интерпретируемость и логичность вывода.

1.4.2 Исследование Spotify — задача регрессии

Аналоги:

– «Spotify Recommender Systems (Spotify API + ML) »;

В реальной практике Spotify использует сложные рекомендательные системы, включая коллаборативную фильтрацию, машинное обучение и нейросети. Для этого используются такие признаки, как danceability, acousticness, energy, valence, tempo и другие метрики аудиоанализов.

Мы использовали часть этих признаков для прогнозирования популярности, что делает наш проект приближённым к реальному кейсу.

– проекты на GitHub.

На платформе GitHub встречаются проекты, использующие датасеты Spotify для предсказания жанра, хит-потенциала или популярности треков.

Многие из них используют похожие признаки, такие как темп, акустичность и инструментальность.

1.4.3 Вывод по проведенному анализу аналогов

Анализ существующих аналогов показал, что задачи классификации и регрессии активно применяются в реальных продуктах и сервисах. Также было подтверждено, что структура нашего проекта соответствует современным практикам в обучении и применении машинного обучения. Выбор задач (Titanic — классификация, Spotify — регрессия) отражает наиболее типовые и прикладные кейсы в области анализа данных.

1.5 Архитектура проекта

1.5.1 Обзор архитектуры программного продукта

Проект представляет собой некий конвейер (pipeline) обработки данных простой линейной архитектуры, состоящий из нескольких ключевых этапов:

1) загрузка и предобработка данных (EDA, очистка, обработка пропусков и выбросов);
2) Feature Engineering (создание новых признаков, масштабирование, кодирование категориальных переменных);
3) анализ и моделирование (выбор и обучение моделей, валидация, интерпретация результатов);
4) визуализация и выводы (графики, отчёты, инсайты).

1.5.2 Описание основных компонентов и связей

Можно выделить такие компоненты:

1) Data Layer (источники данных: CSV, базы данных, API),
2) Processing Layer (Pandas, NumPy для обработки данных),

3) Feature Engineering Layer (библиотеки типа scikit-learn, featuretools),

4) Modeling Layer (ML-алгоритмы, подобранные в ходе исследования),

5) Evaluation & Visualization (Matplotlib/Seaborn, метрики качества).

Связи между ними: данные проходят последовательно через каждый слой, начиная с загрузки и заканчивая выводом результатов.

1.5.3 Обоснование выбора архитектурного решения

Jupyter Notebook выбран из-за удобства интерактивного анализа (быстрая проверка гипотез, визуализация). Библиотеки (Pandas, Scikit-learn) — стандартный стек для Data Science, обеспечивающий скорость разработки. Линейный pipeline (без сложного ветвления) — оправдан, так как задача исследовательская, а не промышленная.

1.6 Методология разработки и процесс анализа данных задачи Titanic

1.6.1 Методология разработки

В рамках проекта по анализу данных о пассажирах «Титаника» и предсказанию их выживаемости была реализована методология, основанная на типичном цикле разработки в области анализа данных и машинного обучения:

1) постановка задачи;

1.1) цель анализа — предсказать факт выживания пассажира на основе его персональных и социальных характеристик;

1.2) подход: задача бинарной классификации (выжил/не выжил) с использованием моделей машинного обучения.

2) сбор и загрузка данных;

2.1) использовался классический датасет Titanic (CSV-файл), содержащий данные о пассажирах: пол, возраст, класс обслуживания, количество родственников на борту, стоимость билета, порт посадки и др.;

2.2) данные загружались с помощью библиотеки pandas.

3) предварительный анализ данных (EDA);

3.1) с помощью визуализаций (matplotlib, seaborn) были изучены распределения признаков и выявлены их связи с переменной выживаемости (Survived) ;

3.2) построены графики: гистограммы, countplot, диаграммы выживаемости по полу, классу, возрасту.;

3.3) проверялись пропущенные значения и аномалии в данных.

4) обработка данных и генерация признаков (Feature Engineering);

4.1) категориальные признаки (пол, порт посадки) были закодированы в числовой формат (Label Encoding / One-Hot Encoding);

4.2) пропущенные значения в признаках, таких как возраст, были заполнены медианой;

4.3) признаки были масштабированы при необходимости (например, возраст, стоимость билета);

4.4) были удалены нерелевантные или избыточные столбцы (например, имя, номер билета).

5) обучение моделей;

5.1) были обучены различные модели классификации: логистическая регрессия, дерево решений, случайный лес, градиентный бустинг, полносвязная нейронная сеть (MLPClassifier);

5.2) производилась настройка гиперпараметров и кросс-валидация (cross_val_score) для оценки стабильности моделей.

6) оценка качества модели;

6.1) использовались метрики качества классификации: accuracy (точность), кросс-валидация;

6.2) сравнение моделей показало, что ансамблевые методы (Random Forest и Gradient Boosting) дали наивысшую точность;

6.3) отдельно была построена нейронная сеть с использованием Keras и TensorFlow, обученная на подготовленных данных и оценённая по точности на тестовой выборке.

1.6.2 Процесс разработки

Процесс разработки состоял из нескольких последовательных этапов, каждый из которых имел чёткую цель и применяемые инструменты:

1) загрузка и первичная обработка данных;

1.1) Датасет был загружен из CSV-файла и прочитан с помощью библиотеки pandas;

1.2) Проведён первичный анализ структуры данных: проверка размерности таблицы, типов данных, наличие пропущенных значений (df.info(), df.describe());

1.3) Определены ключевые категориальные (Sex, Embarked, Pclass) и числовые признаки (Age, Fare, SibSp, Parch) для дальнейшей обработки.

2) исследовательский анализ данных (EDA);

Цель: выявить закономерности, выбросы и зависимые признаки, влияющие на выживаемость.

Выполнены следующие шаги:

2.1) построение распределений признаков (sns.histplot, sns.countplot, sns.boxplot) для анализа структуры и выбросов;

2.2) визуализация взаимосвязей признаков с переменной Survived (например, выживаемость по полу, возрасту, классу) ;

2.3) построение тепловой карты корреляции (sns.heatmap) для оценки линейных взаимосвязей между признаками;

2.4) группировка по ключевым признакам: пол, класс, порт посадки, для анализа долей выживших в разных группах.

Ключевые выводы:

Пол, возраст, класс обслуживания и количество родственников на борту демонстрируют заметное влияние на вероятность выживания.

3) обработка данных и генерация признаков (Feature Engineering);

Цель: подготовить данные к обучению моделей и повысить качество прогнозов.

Выполнены следующие шаги:

3.1) удаление неинформативных признаков (например, Name, Ticket, Cabin);

3.2) заполнение пропусков: Age — медианой, Embarked — модой;

3.3) преобразование категориальных признаков;

3.4) Sex и Embarked были закодированы с помощью LabelEncoder или One-Hot Encoding;

3.5) масштабирование числовых признаков (Age, Fare) с использованием StandardScaler.

4) обучение и валидация моделей;

Цель: построить модель, способную предсказывать факт выживания.

Выполнены следующие шаги:

4.1) разделение данных на обучающую и тестовую выборки (train_test_split);

4.2) использование различных моделей машинного обучения: логистическая регрессия, DecisionTreeClassifier, RandomForestClassifier, GradientBoostingClassifier, Нейронная сеть (MLPClassifier или Keras Sequential);

4.3) проведение перекрёстной проверки (cross_val_score) для оценки устойчивости моделей;

4.4) настройка гиперпараметров с использованием GridSearchCV или ручного подбора.

5) оценка качества моделей;

Использовались метрики классификации:

5.1) Accuracy (доля правильных предсказаний);

5.2) Precision, Recall, F1-score для оценки качества предсказания класса "выживший";

5.3) построение confusion matrix ;

5.4) визуализация качества работы моделей на тестовой выборке.

Вывод: Ансамблевые модели (Random Forest и Gradient Boosting) показали наилучшую точность (до 85%). Также неплохие результаты показала нейронная сеть, обученная на тех же данных.

6) выводы и возможности улучшения.

6.1) некоторые признаки (например, Cabin, Ticket) оказались слабоинформативными и были исключены;

6.2) возможно применение более продвинутых методов обработки категориальных признаков (в том числе эмбеддинги);

6.3) для улучшения качества можно использовать ансамбли моделей, усовершенствовать обработку пропущенных значений и провести более глубокий анализ взаимодействий между признаками.

1.6.3 Заключение по методологии и процессу разработки

Методология разработки и поэтапный процесс реализации проекта обеспечили логичную структуру выполнения работы: от анализа данных до оценки качества моделей. Все шаги реализации, включая подготовку данных, визуализацию, генерацию признаков, обучение моделей и тестирование, представлены в приложении (Приложение А)**,** что позволяет воспроизвести и масштабировать проект в дальнейшем.

1.7 Методология разработки и процесс анализа данных задачи Spotify

1.7.1 Методология разработки

В рамках проекта в задании Spotify по анализу музыкальных треков и предсказанию их популярности была реализована методология, основанная на типичном цикле разработки в области анализа данных и машинного обучения:

1) постановка задачи;

1.1) целью анализа было предсказать уровень популярности музыкальных треков на основе их аудио- и мета-признаков;

1.2) подход: регрессионное моделирование (прогноз значения popularity);

2) сбор и загрузка данных;

2.1) данные были предоставлены в формате CSV и содержали различные аудио-характеристики (такие как energy, danceability, tempo и др.), а также метаданные (жанр, исполнитель и др.).

3) предварительный анализ данных (EDA);

3.1) с помощью визуализации (гистограммы, диаграммы рассеяния, тепловые карты корреляции) были выявлены закономерности в данных;

3.2) Проверялись пропущенные значения, распределения признаков и их связь с целевой переменной.

4) обработка данных и генерация признаков (Feature Engineering);

4.1) признаки были масштабированы и нормализованы при необходимости;

4.2) категориальные признаки закодированы (например, one-hot-кодированием);

4.3) были созданы новые признаки на основе имеющихся (например, логарифм темпа, индикаторы жанра и т.д.).

5) обучение моделей;

5.1) были опробованы различные модели машинного обучения (например, линейная регрессия, деревья решений, ансамблевые методы);

5.2) производилась настройка гиперпараметров с помощью кросс-валидации.

6) оценка качества модели.

6.1) использовались метрики качества для регрессии: MAE, MSE, RMSE, R^2;

6.2) модель проверялась на валидационных и тестовых выборках для оценки обобщающей способности.

1.7.2 Процесс разработки

Процесс разработки состоял из нескольких последовательных этапов, каждый из которых имел чёткую цель и инструменты реализации:

1) загрузка и первичная обработка данных;

1.1) датасет был загружен из CSV-файла и прочитан с помощью библиотеки pandas;

1.2) проверялись размеры таблицы, типы данных, наличие пропущенных значений (df.info(), df.describe());

1.3) категориальные и числовые признаки были выделены в отдельные списки для удобства дальнейшей обработки.

2) исследовательский анализ данных (EDA);

Цель — выявить закономерности, тренды, выбросы и зависимости между признаками.

Были выполнены:

2.1) построение распределений признаков (sns.histplot, sns.boxplot);

2.2) построение тепловой карты корреляции для поиска сильных зависимостей;

2.3) исследование зависимости popularity от других признаков— таких как danceability, energy, tempo, duration_ms;

2.4) группировка данных по жанрам, исполнителям и анализ средней популярности.

Ключевые выводы:

Признаки danceability, energy, valence, acousticness имеют умеренную корреляцию с популярностью.

3) обработка данных и генерация признаков (Feature Engineering);

Цель — подготовить данные к обучению модели и повысить её качество.

Были предприняты следующие действия:

3.1) удаление ненужных признаков (например, названия треков, track_id, time_signature, mode — неинформативны);

3.2) преобразование категориальных признаков (genre, artist_name) с помощью one-hot-кодирования;

3.3) масштабирование числовых признаков с помощью MinMaxScaler или StandardScaler.

4) обучение и валидация моделей;

Цель — построить модель, способную точно предсказывать популярность трека.

Были использованы:

4.1) разделение на обучающую и тестовую выборки: train_test_split();

4.2) модели: Linear Regression, DecisionTreeRegressor, RandomForestRegressor, GradientBoostingRegressor;

4.3) кросс-валидация (cross_val_score) для оценки стабильности модели;

4.4) настройка гиперпараметров через GridSearchCV.

5) оценка качества модели;

Использовались метрики:

5.1) MAE (Mean Absolute Error);

5.2) MSE (Mean Squared Error);

5.3) R^2 (коэффициент детерминации).

Вывод:

Лучшую производительность показали ансамблевые модели, особенно Random Forest и Gradient Boosting.R^2 для лучших моделей превышал 0.70, что свидетельствует о хорошем качестве прогноза.

6) выводы и возможности улучшения.

6.1) некоторые признаки оказали слабое влияние на популярность и могут быть исключены;

6.2) влияние жанра и исполнителя выражено через one-hot-кодирование, но возможно использовать эмбеддинги;

6.3) возможны дальнейшие улучшения за счёт нейросетевых моделей и анализа текста (например, названий треков).

1.8 Заключение по методологии и процессу разработки

Методология разработки (см. раздел 1.7.1) и пошаговый процесс реализации (см. раздел 1.7.2) обеспечили системный подход к построению модели и её интерпретации. Все этапы проекта — от подготовки данных до оценки модели — подробно реализованы в коде, приведённом в приложении (Приложение В), что обеспечивает воспроизводимость и даёт возможность масштабирования проекта в будущем.

1.9 Тестирование и выявленные ошибки

В ходе работы над соревнованием по Титанику проводилось поэтапное тестирование различных подходов к прогнозированию выживаемости на датасете.

Нейронная сеть показала умеренную эффективность из-за малого объёма данных. На таком масштабе сложные архитектуры не дали значительного преимущества перед классическими методами ML. Random Forest и Gradient Boosting показали более высокую точность при меньших вычислительных затратах. Гиперпараметрическая оптимизация

(GridSearchCV, RandomizedSearchCV) дала заметный прирост качества для этих моделей.

Таким образом для задач с небольшим объёмом данных нейросети не всегда оптимальны. Лучше применять классические методы ML, и основные усилия по улучшению модели должны быть направлены на Feature Engineering и подбор гиперпараметров, а не на усложнение архитектуры.

В ходе работы над соревнованием по Spotify проводилось последовательное тестирование двух моделей машинного обучения: линейной регрессии и градиентного бустинга.

Линейная регрессия является достаточно эффективным решением для данной задачи, это скорее исключение из правил и может говорить о достаточности линейных зависимостей для данной задачи, а также возможной избыточности сложных моделей, таких как градиентный бустинг.

Градиентный бустинг показал ожидаемо хорошие результаты, но требует дополнительной настройки для раскрытия своего потенциала. Так желательно провести систематический подбор гиперпараметров, поэкспериментировать с различными функциями потерь, рассмотреть ансамбли с другими моделями.

1.10 Планирование и распределение задач

Проект разрабатывался по гибкой методологии с постоянными контактами через мессенджеры для синхронизации. Один член команды выполнял роль аналитика данных, другой – ML-инженера. При трудностях в выполнении задач сроки корректировались путем перераспределения нагрузки.

Такой подход обеспечил согласованность действий, эффективное разделение обязанностей и повысил продуктивность при выполнении учебного проекта.

ЗАКЛЮЧЕНИЕ

Оценка соответствия требованиям заказчика и пользователя

Программный продукт, состоящий из двух аналитических блоков — анализа выживаемости пассажиров «Titanic» и музыкальных предпочтений пользователей Spotify — соответствует поставленным задачам. Основные цели были направлены на:

– разработку моделей машинного обучения,

– проведение разведочного анализа данных (EDA),

– интерпретацию результатов для дальнейшего принятия решений.

Исследование Titanic решает задачу классификации на основе реальных данных, демонстрируя грамотное использование методов предобработки, one-hot-кодирования, масштабирования и кросс-валидации. Исследование Spotify позволяет провести кластеризацию музыкальных треков и выявить предпочтения пользователей, что соответствует пользовательскому сценарию анализа вкусов.

Таким образом, поставленные задачи не просто решены, но и реализованы с акцентом на практическую применимость моделей, что говорит о соответствии ожиданиям пользователей и учебным целям.

Оценка качества и выявление проблем

Качество программного продукта можно оценить по следующим аспектам:

– результаты тестирования моделей показывают адекватную точность и стабильность. Модель Titanic достигает высоких метрик (точность, recall, f1-score) без признаков переобучения, что говорит о хорошей обобщающей способности;

– в проекте Spotify кластеризация KMeans была валидирована визуализацией, а разбиение треков по жанрам/группам логично и интерпретируемо.

Выявленные проблемы:

– в проекте Titanic встречались пропуски в данных (например, в признаках «Age» и «Cabin»), которые потребовали аккуратной обработки, что потенциально могло повлиять на стабильность модели;

– в Spotify возможно переусложнение модели при выборе количества кластеров.

Предложения по улучшению и развитию продукта

Для улучшения качества и расширения функционала продукта предлагаются следующие шаги:

1) модульность кода: вынести повторяющийся код в функции/классы для повышения читаемости и повторного использования;
2) повышение качества визуализаций: добавить аннотации, легенды и пояснения на графиках, чтобы обеспечить полноту интерпретации;
3) автоматизация: интегрировать тестирование (например, с использованием pytest для функций обработки данных) для контроля корректности на всех этапах;
4) расширение функционала Spotify-анализа:

4.1) подключение внешних API (например, Spotify API) для анализа в реальном времени;

4.2) добавление пользовательских фильтров и генерации рекомендаций.

5) интерфейс пользователя: возможна реализация простого веб-интерфейса (например, с использованием Streamlit или Flask) для нетехнических пользователей.

Вывод

Программный продукт достиг своих основных целей и продемонстрировал работоспособность и практическую применимость. Оценка моделей и результатов обработки данных подтверждает обоснованность принятых решений. Однако возможны улучшения по части

структуры кода, автоматизации и пользовательского интерфейса, что открывает путь к дальнейшему развитию проекта как полноценного аналитического инструмента.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лутц М. Изучаем Python. Пер. с англ. 5-е изд. — СПб.: Питер, 2021. — 1216 с.

2. Герасимов А., Демидов Д. Машинное обучение. Курс лекций [Электронный ресурс] // Школа анализа данных Яндекса. — URL: https://mlcourse.ai (дата обращения: 25.05.2025).

3. Raschka S., Mirjalili V. Python и машинное обучение. Подробный справочник по методам машинного обучения и глубокого обучения с использованием scikit-learn, Keras и TensorFlow. — М.: Диалектика, 2020. — 816 с.

4. Ковалевская Е.Ю., Беляев С.В. Основы машинного обучения. Учебное пособие. — М.: МГТУ им. Н.Э. Баумана, 2021. — 224 с.

5. Géron A. Прикладное машинное обучение с использованием Scikit-Learn, Keras и TensorFlow. Концепции, инструменты и методы для создания интеллектуальных систем. — М.: ДМК Пресс, 2020. — 848 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода проекта Titanic

```
#библиотека для работы с таблицами
import pandas as pd
#библиотека для числовых вычислений
import numpy as np
# Импортируем модуль для создания легенды вручную
import matplotlib.patches as mpatches
#для построения графиков
import matplotlib.pyplot as plt
#Для красивой визуализации данных на основе Matplotlib
import seaborn as sns
#Загружаем CSV-файл с данными о пассажирах Титаника.
#df = pd.read_csv("Titanic.csv")
#Анализ первичных данных
df.info()
df.describe()
# Визуализация пропусков
# Устанавливаем размер
plt.figure(figsize=(20, 20))
#Возвращение отстутсвующих значений( df.isnull() возвращает таб-
лицу того же размера, где: True —  значение отсутствует (NaN);
#sns.heatmap(...) строит тепловую карту, где:True (NaN) — отобра-
жается как цвет ( оттенки красного из палитры 'Reds');
# cbar=False — отключает цветовую шкалу сбоку
sns.heatmap(df.isnull(), cbar=False, cmap='Reds')
# Добавляем заголовок к графику
plt.title("Отсутствующие значения")
# Отображаем график
plt.show()

#Распределение по выживаемости
```

```
# Устанавливаем тип столбчатой диаграммы из библиотеки seaborn.
# x='Survived' — мы строим график по колонке Survived(0 — пассажир не выжил, 1 — пассажир выжил.)
sns.countplot(data=df, x='Survived')
# Добавляем заголовок к графику
plt.title("Распределение по выживаемости")
# Добавляем метку для оси X
plt.xlabel('Выживаемость')
# Добавляем метку для оси Y
plt.ylabel('Количество пассажиров')
# Отображаем график
plt.show()
#Пол и выживаемость
# Устанавливаем размер
plt.figure(figsize=(10, 6))
# Устанавливаем тип столбчатой диаграммы из библиотеки seaborn.
# распределение количества пассажиров по полу (Sex), разделяя каждый пол по признаку выживания (Survived).
sns.countplot(data=df, x='Sex', hue='Survived', palette='muted')
# Добавляем заголовок к графику
plt.title("Пол и выживаемость")
# Добавляем метку для оси X
plt.xlabel('Зависимость выживаемости от пола')
# Добавляем метку для оси Y
plt.ylabel('Количество пассажиров')
# Добавляем легенду
# title - задает заголовок легенды
# title_fontsize - размер шрифта заголовка легенды
plt.legend(title='Выживаемость (0 -количество невыживших, 1-количество выживших)', title_fontsize=10)
plt.show()
#Класс и выживаемость
# Устанавливаем размер
plt.figure(figsize=(10, 6))
# Устанавливаем тип столбчатой диаграммы из библиотеки seaborn.
```

```
# Строим столбчатую диаграмму: сравнение классов (Pclass) и выживаемости (Survived)
sns.countplot(data=df, x='Pclass', hue='Survived', palette='muted')
# Добавляем заголовок к графику
plt.title("Класс и выживаемость")
# Добавляем метку для оси X
plt.xlabel("Класс пассажира (Pclass)")
# Добавляем метку для оси Y
plt.ylabel("Количество пассажиров")
# Добавляем легенду
# title - задает заголовок легенды
# title_fontsize - размер шрифта заголовка легенды
plt.legend(title='Выживаемость (0 -количество невыживших, 1-количество выживших)', title_fontsize=10)
plt.show()
#Возраст и выживаемость
# Устанавливаем размер
plt.figure(figsize=(10, 10))
# Устанавливаем тип столбчатой диаграммы из библиотеки seaborn.
# x='Age' — по оси X отложены значения возраста.
#hue='Survived' — окрашиваем столбцы в зависимости от выживаемости (0 -количество невыживших, 1-количество выживших).
#bins=30 — делим возрастной диапазон на 30 интервалов.
#kde=True — добавляем линию плотности распределения
#palette='Set2' — используем готовую цветовую палитру Set2 для раскраски по Survived (зеленый(0)-количество невыживших, оранжевый(1)-количество выживших.)
#также не забываем отфильтровать строки с пропущенными значениями в нужной колонке перед построением графика
#df[df['Age'].notna()] — фильтрует только строки, где Age заполнен.
sns.histplot(data=df[df['Age'].notna()], x='Age', hue='Survived', bins=30, kde=True, palette='Set2')
# Добавляем заголовок к графику
```

```
plt.title("Возраст и выживаемость")
# Добавляем метку для оси X
plt.xlabel("Возраст пассажира")
# Добавляем метку для оси Y
plt.ylabel("Количество пассажиров")
plt.show()
#Статус пассажира и выживаемость
# Расчет доли выживших по обращениям
# .str.extract(r' ([A-Za-z]+)\.', expand=False) ищет обращение (Mr, Mrs, Dr) в строке Name и сохраняет его в новую колонку Title
df['Title'] = df['Name'].str.extract(r' ([A-Za-z]+)\.', expand=False)
#df.groupby('Title') — группирует строки датафрейма по значению в колонке Title .
#['Survived'].mean() — для каждой группы считает среднее значение по колонке Survived.
# .sort_values(ascending=False) — сортирует результат по убыванию
title_survival = df.groupby('Title')['Survived'].mean().sort_values(ascending=False).reset_index()
# Устанавливаем размер
plt.figure(figsize=(12, 10))
#Устанавливаем тип столбчатой диаграммы из библиотеки seabor
#x='Title' — категории по оси X (типы обращений).
#y='Survived' — высота столбца соответствует доле выживших для каждого титула.
sns.barplot(data=title_survival, x='Title', y='Survived')
# Добавляем заголовок к графику
plt.title("Доля выживших по обращениям (Title)")
# Добавляем метку для оси X
plt.xlabel("Обращение (Title)")
# Добавляем метку для оси Y
plt.ylabel("Доля выживших")
# Поворачивает подписи на оси X на 45°
plt.xticks(rotation=45)
plt.show()
```

```
#Количество родственников (SibSp + Parch) и выживаемость
# Создаём новый столбец FamilySize, складывая количество братьев/сестёр/супругов (SibSp) и родителей/детей (Parch)
df['FamilySize'] = df['SibSp'] + df['Parch']
# Устанавливаем размер
plt.figure(figsize=(10, 6))
# Устанавливаем тип столбчатой диаграммы из библиотеки seaborn: по оси X — размер семьи, по цвету (hue) — выживаемость
sns.countplot(data=df, x='FamilySize', hue='Survived', palette='muted')
# Добавляем заголовок к графику
plt.title("Размер семьи и выживаемость")
# Добавляем метку для оси X
plt.xlabel("Количество родственников на борту")
# # Добавляем метку для оси Y
plt.ylabel("Число пассажиров")
# Добавляем легенду
# title - задает заголовок легенды
# title_fontsize - размер шрифта заголовка легенды
plt.legend(title='Выживаемость (0 -количество невыживших, 1-количество выживших)', title_fontsize=10)
# Отображаем график
plt.show()
#Корреляция признаков
#df.corr(numeric_only=True) — вычисляется корреляционная матрица только по числовым столбцам.
#sns.heatmap(...) — строим тепловую карту (heatmap) с помощью библиотеки seaborn.
#annot=True — добавляет численные значения корреляции на график.
#cmap='coolwarm' — цветовая палитра: красные оттенки для отрицательной корреляции, синие — для положительной.
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm')
# Добавляем заголовок
plt.title("Корреляция признаков")
```

```
plt.show()
#Импорт необходимых библиотек
import numpy as np
import pandas as pd
import os
from sklearn import tree
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from IPython.display import display
titanic = pd.read_csv("C:\\Users\\drozd\\DataScience\\Titanic\\Titanic.csv")
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense, Dropout, Input
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
#Предобработка данных (Data Preprocessing) — это подготовка "сырых" данных к анализу и машинному обучению.
titanic.isna().sum()
#Обработка пропущенных значений:
# Заполнение пропусков в возрастах медианным значением
titanic['Age'] = titanic['Age'].fillna(titanic['Age'].median())
# Заполнение пропусков в столбце 'Embarked' значением самой популярной записи
titanic['Embarked'] = titanic['Embarked'].fillna(titanic['Embarked'].mode()[0])
#Вывод всех доступных столбцов в DataFrame Titanic для кодирования категориальных данных
print(titanic.columns)
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

```
# Извлекаем 'Title' из 'Name'
titanic['Title'] = titanic['Name'].str.extract(' ([A-Za-z]+)\.',
expand=False)
titanic['Title'] = titanic['Title'].fillna('Unknown')  # Заполняем
пропуски
# Кодирование категориальных данных
dan_features = ['Sex', 'Embarked', 'Title']
# Кодируем категориальные признаки
for feature in dan_features:
    label_encoder = LabelEncoder()
    titanic[feature]  =  label_encoder.fit_transform(titanic[fea-
ture])
# Удаляем ненужные столбцы
titanic.drop(['Name', 'Ticket', 'Cabin', 'PassengerId'], axis=1,
inplace=True)
#Подготовка данных для обучения:
# Удаляем столбец 'Выживший' из X (признаки) и сохраняем его в y
(метки)
X = titanic.drop('Survived', axis=1)  # Все колонки, кроме 'Вы-
живший'
y = titanic['Survived']  # Только колонка 'Выживший', которая со-
держит метки классов
# Масштабирование данных
# Создаем экземпляр стандартного масштабатора
scaler = StandardScaler()
# Применяем масштабирование к данным (нормализация признаков)
X = scaler.fit_transform(X)
# Преобразование в массивы numpy
X = np.array(X)
y = np.array(y)
# Разделение данных на обучающую и тестовую выборки
# Разделяем данные на обучающую и тестовую выборки (80% для обу-
чения, 20% для тестирования)
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,
test_size=0.2, random_state=50)
```

```
# Преобразование меток в one-hot кодировку
# Преобразуем метки классов в бинарную (one-hot) кодировку для нейросетей
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
Создание и обучение модели нейронной сетиfrom sklearn.model_selection import KFold
from sklearn.metrics import classification_report, confusion_matrix
from keras.layers import Dense, Input
from keras.callbacks import EarlyStopping
from keras.utils import to_categorical
#Настройка 5-кратной кросс-валидации
# Установим параметры кросс-валидации
kf = KFold(n_splits=5, shuffle=True, random_state=50)
# Имена целевых классов для отчетов
target_names = ['Died', 'Survived']
# Списки для хранения результатов каждой итерации
all_reports = []  # Для хранения отчетов классификации
all_confusion_matrices = []  # Для хранения матриц ошибок
all_accuracies = []  # Для хранения точностей
all_losses = []  # Для хранения значений потерь
all_train_accuracies = []
all_train_losses = []
# Установка ранней остановки для предотвращения переобучения
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
# Проход по каждому фолду в кросс-валидации
for train_index, val_index in kf.split(X):
    # Разделяем данные на обучающую и валидационную выборки
    X_train_fold, X_val_fold = X[train_index], X[val_index]
    y_train_fold, y_val_fold = y[train_index], y[val_index]
    # Преобразуем метки в формат one-hot для многоклассовой классификации
```

```
    y_train_fold = to_categorical(y_train_fold)  # Преобразование меток в one-hot
    y_val_fold = to_categorical(y_val_fold)
    # Определяем размерность входных данных
    input_dim = X_train_fold.shape[1]

    # Создаем модель нейронной сети
    model = Sequential()
    model.add(Input(shape=(input_dim,)))  # Определяем форму входных данных
    model.add(Dense(80, activation='relu'))  # Первый скрытый слой с 80 нейронами
    model.add(Dense(2, activation='softmax'))  # Выходной слой для 2 классов
    # Компилируем модель с выбранной функцией потерь и оптимизатором
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    # Обучаем модель на обучающей выборке с валидацией
    history = model.fit(
        X_train_fold, y_train_fold,
        validation_data=(X_val_fold, y_val_fold),
        epochs=50, batch_size=40, verbose=0)
    # Получаем предсказания модели на валидационном наборе
    y_pred_fold = model.predict(X_val_fold)
    y_pred_classes = np.argmax(y_pred_fold, axis=1)  # Предсказанные классы
    y_true_classes = np.argmax(y_val_fold, axis=1)  # Истинные классы
    # Сохраняем значения точности и потерь для каждой итерации
    all_train_accuracies.append(history.history['accuracy'])
    all_train_losses.append(history.history['loss'])
    all_accuracies.append(history.history['val_accuracy'])
    all_losses.append(history.history['val_loss'])
    # Генерируем отчет о классификации и матрицу ошибок
```

```
    report = classification_report(y_true_classes, y_pred_classes, target_names=target_names)
    confusion = confusion_matrix(y_true_classes, y_pred_classes)
    # Сохраняем результаты отчетов и матрицы ошибок
    all_reports.append(report)
    all_confusion_matrices.append(confusion)
рафик Bar Plot по метрикам для каждого фолда
# Сбор метрик для каждого фолда
fold_metrics = []
for i, (report, cm) in enumerate(zip(all_reports, all_confusion_matrices)):
    tn, fp, fn, tp = cm.ravel()
    accuracy = (tp + tn) / (tp + tn + fp + fn)
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0
    f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
    fold_metrics.append({
        'Fold': f'Fold {i+1}',
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-Score': f1
    })
# Преобразуем в DataFrame
metrics_df = pd.DataFrame(fold_metrics)
# Визуализация
plt.figure(figsize=(12, 6))
sns.barplot(data=pd.melt(metrics_df, id_vars='Fold'), x='Fold', y='value', hue='variable')
plt.title('Метрики классификации по фолдам')
plt.ylabel('Значение')
plt.ylim(0, 1)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
```

```
plt.show()
# Усредненные значения по фолдам
mean_accuracy = np.mean(all_accuracies, axis=0)
mean_loss = np.mean(all_losses, axis=0)
# Стандартные отклонения
std_accuracy = np.std(all_accuracies, axis=0)
std_loss = np.std(all_losses, axis=0)
# Диапазон эпох
epochs = range(1, len(mean_accuracy) + 1)
plt.figure(figsize=(14, 6))
# График точности
plt.subplot(1, 2, 1)
plt.plot(epochs, mean_accuracy, 'b-', label='Средняя точность (ва-
лидация)')
plt.fill_between(epochs,
                 mean_accuracy - std_accuracy,
                 mean_accuracy + std_accuracy,
                 color='blue', alpha=0.2, label='±1 std')
plt.xlabel('Эпохи')
plt.ylabel('Точность')
plt.title('Точность по фолдам с доверительным интервалом')
plt.legend()
plt.grid(True)
# График потерь
plt.subplot(1, 2, 2)
plt.plot(epochs, mean_loss, 'r-', label='Средние потери (валида-
ция)')
plt.fill_between(epochs,
                 mean_loss - std_loss,
                 mean_loss + std_loss,
                 color='red', alpha=0.2, label='±1 std')
plt.xlabel('Эпохи')
plt.ylabel('Потери')
plt.title('Потери по фолдам с доверительным интервалом')
plt.legend()
```

```
plt.grid(True)

plt.tight_layout()
plt.show()

# График точности (тренировка + валидация)
plt.subplot(1, 2, 1)
plt.plot(epochs, mean_train_accuracy, 'g-', label='Точность
(тренировка)')
plt.plot(epochs, mean_accuracy, 'b-', label='Точность
(валидация)')
plt.xlabel('Эпохи')
plt.ylabel('Точность')
plt.title('Сравнение точности на тренировке и валидации')
plt.legend()
plt.grid(True)

# График потерь (тренировка + валидация)
plt.subplot(1, 2, 2)
plt.plot(epochs, mean_train_loss, 'orange', label='Потери
(тренировка)')
plt.plot(epochs, mean_loss, 'r-', label='Потери (валидация)')
plt.xlabel('Эпохи')
plt.ylabel('Потери')
plt.title('Сравнение потерь на тренировке и валидации')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
# Печатаем отчет о классификации и матрицу ошибок для каждого фолда
в текстовом виде
for i, (report, cm) in enumerate(zip(all_reports, all_confu-
sion_matrices)):
    print(f"\nФолд {i+1}:\n")
```

```
    print("Отчет о классификации:")
    print(report)
    print("\nМатрица ошибок:")
    print(cm)
    print("\n" + "="*50 + "\n")

# Выводим итоговую матрицу ошибок в текстовом виде
total_confusion_matrix = np.sum(all_confusion_matrices, axis=0)
print("\nИтоговая матрица ошибок по всем фолдам:")
print(total_confusion_matrix)
```

ПРИЛОЖЕНИЕ В

(обязательное)

Листинг кода проекта Spotify

```
Импорт библиотек
#библиотека для работы с таблицами
import pandas as pd
#библиотека для числовых вычислений
import numpy as np
#для построения графиков
import matplotlib.pyplot as plt
#Для красивой визуализации данных на основе Matplotlib
import seaborn as sns
Загружаем CSV-файл с данными
In [154]:
# Загрузка данных
sf = pd.read_csv("dataset.csv")
sf.head()
#Анализ первичных данных
# Проверка на пропуски
sf.info()
sf.isnull().sum()
sf.describe()
#Визуализация пропусков
# Устанавливаем размер
plt.figure(figsize=(20, 20))
#Возвращение отстутсвующих значений( sf.isnull() возвращает таб-
лицу того же размера, где: True —  значение отсутствует (NaN);
#sns.heatmap(...) строит тепловую карту, где:True (NaN) — отобра-
жается как цвет ( оттенки красного из палитры 'Reds');
# cbar=False — отключает цветовую шкалу сбоку
sns.heatmap(sf.isnull(), cbar=False, cmap='Reds')
# Добавляем заголовок к графику
plt.title("Отсутствующие значения")
# Отображаем график
plt.show()
# Распределение таргетной переменной
```

```
# Устанавливаем размер
plt.figure(figsize=(10, 5))
# Построение гистограммы  с помощью библиотеки Seaborn.
#sf['popularity'] — берет столбец popularity из sf
##bins=30 — количество столбцов
#kde=True — включает линию оценки плотности распределения
#color='skyblue' —  цвет гистограммы (небесно-голубой)
sns.histplot(sf['popularity'],  bins=30,  kde=True,  color='sky-
blue')
# Добавляем заголовок к графику
plt.title('Распределение популярности треков')
# Добавляем метку для оси X
plt.xlabel('Популярность')
# Добавляем метку для оси Y
plt.ylabel('Количество треков')
# Отображаем график
plt.show()
```

Распределение популярности по жанрам

```
# Группируем по жанру  и считаем среднюю популярность
genre_popularity      =      sf.groupby('track_genre')['popularі-
ty'].mean().sort_values(ascending=False).reset_index()
# Устанавливаем размер
plt.figure(figsize=(14, 6))
#Создаётся график типа barplot .
#hue='track_genre' —  каждый столбик  окрашен в свой цвет в зави-
симости от жанра.
sns.barplot(data=genre_popularity,  x='track_genre',  y='popularі-
ty',hue='track_genre', )
# Добавляем заголовок к графику
plt.title("Средняя популярность по жанрам треков")
# Добавляем метку для оси X
plt.xlabel("Жанр")
# Добавляем метку для оси Y
plt.ylabel("Средняя популярность")
# Поворачиваеv подписи на оси X на 90°
```

```
plt.xticks(rotation=90)
# Отображаем график
plt.show()
# Группируем по жанру и считаем среднюю популярность
genre_popularity = sf.groupby('track_genre')['popularity'].mean().sort_values(ascending=False).reset_index()
# Отбираем топ-10 жанров
top_10_genres = genre_popularity.head(10)
plt.figure(figsize=(12, 6))
sns.barplot(data=top_10_genres, x='track_genre',y='popularity',hue='track_genre', palette='magma',dodge=False )
plt.title("Топ-10 жанров по средней популярности треков")
plt.xlabel("Жанр")
plt.ylabel("Средняя популярность")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
#Влияние наличия нецензурной лексики на популярность треков
#устанавливаем размер
plt.figure(figsize=(8, 5))
# x='popularity'- Значения по оси X
# hue='explicit' - Цветовая группировка по признаку 'explicit'
# multiple='stack'- Столбики с разными цветами складываются друг на друга
# bins=20-Разбиение данных на 20 интервалов
# palette='coolwarm' - Цветовая палитра: контрастные оттенки для двух групп
sns.histplot(data=sf, x='popularity', hue='explicit', multiple='stack', bins=20, palette='coolwarm')
# Добавляем заголовок к графику
plt.title('Популярность треков в зависимости от наличия нецензурной лексики')
# Добавляем метку для оси X
plt.xlabel('Популярность')
# Добавляем метку для оси Y
```

```
plt.ylabel('Количество треков')
# Отображаем график
plt.show()
#Влияние длительности трека на популярность
# Переводим длительность трека из миллисекунд в минуты.
sf['duration_min'] = sf['duration_ms'] / 60000
# Группируем по диапазонам длительности
bins = [0, 2, 3, 4, 5, 10, sf['duration_min'].max()]
labels = ['<2 мин', '2-3 мин', '3-4 мин', '4-5 мин', '5-10 мин',
'>10 мин']
# группируем по длине
sf['duration_group'] = pd.cut(sf['duration_min'], bins=bins, la-
bels=labels, include_lowest=True)
# Boxplot популярности по группам длительности
plt.figure(figsize=(10, 6))
sns.boxplot(data=sf, x='duration_group', y='popularity')
# Добавляем заголовок к графику
plt.title('Распределение популярности по группам длительности тре-
ка', fontsize=14)
# Добавляем метку для оси X
plt.xlabel('Длительность трека', fontsize=12)
# Добавляем метку для оси Y
plt.ylabel('Популярность', fontsize=12)
plt.grid(axis='y')
# Отображаем график
plt.show()
plt.figure(figsize=(8, 5))
# Гистограмма длительности
sns.histplot(sf['duration_min'], bins=60, kde=True, color='sky-
blue')
plt.xlim(0, 10)
# Добавляем заголовок к графику
plt.title('Распределение длительности треков (минуты)',
fontsize=14)
# Добавляем метку для оси X
```

```
plt.xlabel('Длительность трека (мин)', fontsize=12)
# Добавляем метку для оси Y
plt.ylabel('Количество треков', fontsize=12)
#включаем сетку о по оси Y
plt.grid(axis='y')
# Отображаем график
plt.show()
#Влияние исполнителя на популярность
plt.figure(figsize=(10, 6))
sns.barplot(
    x=artist_popularity.values, # Значения средней популярности артистов
    y=artist_popularity.index,  # Имена артистов
    hue=artist_popularity.index,  # используем сами имена артистов как hue
    palette='mako', #цветовая палитру 'mako'

)
# Добавляем заголовок к графику
plt.title('Средняя популярность треков у топ-10 артистов', fontsize=14)
plt.xlabel('Средняя популярность')
plt.ylabel('Артист')
#включаем сетку о по оси X
plt.grid(axis='x')
#Влияние альбома на популярность треков
#Группируем данные по названиям альбомов (album_name).Для каждого альбома считаем среднее значение популярности треков (popularity).
#Сортируем альбомы по средней популярности в порядке убывания.#Берём топ-15 самых популярных альбомов.
album_popularity = sf.groupby('album_name')['popularity'].mean().sort_values(ascending=False).head(15)
#Задаём размер
plt.figure(figsize=(10, 6))
```

```
sns.barplot(x=album_popularity.values,  y=album_popularity.index,
color='#5A9')
plt.title('Топ-15 альбомов по средней популярности треков')
plt.xlabel('Средняя популярность')
plt.ylabel('Название альбома')
plt.tight_layout()
plt.show()
#Зависимость популярности от танцевальности
# 4. Гистограмма распределения танцевальности в зависимости от
популярности
plt.figure(figsize=(10, 5))
#sns.scatterplot — строит диаграмму рассеяния (scatter plot).
alpha=0.3 — устанавливаем прозрачность точек
sns.scatterplot(data=sf,  x='danceability',  y='popularity',  al-
pha=0.3)
plt.title('Зависимость популярности от танцевальности')
plt.xlabel('Танцевальность')
plt.ylabel('Популярность')
plt.show()
#Зависимость популярности от энергетичности
# 5. Гистограмма: распределение энергетичности и популярности
plt.figure(figsize=(10, 5))
sns.scatterplot(data=sf,  x='energy',  y='popularity',  alpha=0.3,
color='green')
plt.title('Зависимость популярности от энергетичности')
plt.xlabel('Энергия')
plt.ylabel('Популярность')
plt.show()
#Корреляция признаков
# Создаем копию датафрейма для преобразования
df_encoded = sf.copy()
# Кодируем все нечисловые признаки в числовые (типа object,
category и  bool) в числовой формат с помощью функции pd.factorize.
for col in df_encoded.columns:
```

```
    if df_encoded[col].dtype == 'object' or df_encoded[col].dtype.name == 'category' or df_encoded[col].dtype == 'bool':
        df_encoded[col], _ = pd.factorize(df_encoded[col])

#  вычисляем корреляционную матрицу по всем признакам
corr_matrix = df_encoded.corr()
# Визуализация корреляционной матрицы
plt.figure(figsize=(14, 10))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Корреляционная матрица всех признаков ', fontsize=16)
plt.show()

#sf.corr(numeric_only=True) — вычисляется корреляционная матрица только по числовым столбцам.
#sns.heatmap(...) — строим тепловую карту (heatmap) с помощью библиотеки seaborn.
#annot=True — добавляет численные значения корреляции на график.
#fmt=".2f" — форматирует числа с двумя знаками после запятой.
#cmap='coolwarm' — цветовая палитра: красные оттенки для отрицательной корреляции, синие — для положительной.
plt.figure(figsize=(12, 8))
sns.heatmap(sf.corr(numeric_only=True), annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Корреляционная матрица")
plt.show()
Feature Engineering
#Загрузка исходного датасета
#data = pd.read_csv("dataset.csv")
#Импорт необходимых библиотек
import numpy as np
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_absolute_error, mean_squared_er-
ror
from sklearn.neighbors import KNeighborsRegressor
from sklearn.inspection import permutation_importance
# Применяем функцию int к каждому элементу в колонке 'explicit'
data.explicit = data.explicit.apply(int)
Категориальные и числовые признаки
# Получаем список категориальных столбцов (объекты и категории) из
DataFrame
cat_cols    =    data.select_dtypes(include=['object',    'catego-
ry']).columns.to_list()
# Получаем список числовых столбцов из DataFrame
num_cols      =      data.select_dtypes(include=['number']).col-
umns.to_list()
# График корреляции числовых признаков
plt.figure(figsize=(12, 10))
corr_matrix = data[num_cols].corr()
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm',
            mask=np.triu(corr_matrix), vmin=-1, vmax=1)
plt.title('Корреляция числовых признаков')
plt.show()
# Получаем количество уникальных значений для каждого числового
столбца
num_cols_nunique = data[num_cols].nunique()
# Выводим количество уникальных значений для числовых столбцов
num_cols_nunique
#Предобработка данных
```

```
# Добавляем категории 'explicit', 'key', 'mode' и 'time_signature'
в список категориальных столбцов
cat_cols.extend(['explicit', 'key', 'mode', 'time_signature'])
# Удаляем категории 'explicit', 'key', 'mode' и 'time_signature'
из списка числовых столбцов
for item in ['explicit', 'key', 'mode', 'time_signature']:
    num_cols.remove(item)
# Выводим обновлённый список категориальных столбцов
cat_cols
#Проверим числовые признаки на отклонение значений
data[num_cols].skew().sort_values()
#Преобразуем данные дальше:
#Добавляем новые данные
#Преобразуем категориальные данные
# Удаляем ненужные столбцы из DataFrame
data = data.drop([ 'track_id', 'Unnamed: 0',
                  'duration_ms', 'track_name', 'artists', 'al-
bum_name', 'key'], axis=1)

# Создаём новый столбец 'dance_energy_rat' как отношение
'danceability' к 'energy', избегая деления на ноль
data['dance_energy_rat'] = data['danceability'] / (data['energy']
+ 1e-6)
# Отображаем первые 5 строк обновлённого DataFrame
data.head()
#Посмотрим на кол-во уникальных значений, чтобы определить, какое
применить кодирование
data.nunique()
# One-Hot Encoding (OHE)
data = pd.get_dummies(data, columns=['mode', 'time_signature',
'explicit'], drop_first = True)
# Target Encoding
data['category_track_genre'] = data.groupby('track_genre')['popu-
larity'].transform('mean')
```

Создаем признаки и указываем таргетную переменную

```
# Получаем список всех столбцов в DataFrame и сохраняем его в переменной 'features'
features = data.columns.to_list()
# Удаляем столбец 'popularity' из списка признаков, так как он будет использоваться как целевая переменная
features.remove('popularity')
features.remove('track_genre')
#Создание моделей машинного обучения
#Импорт необходимых библиотек
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, make_scorer
import numpy as np
from sklearn.model_selection import GridSearchCV, train_test_split, cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_er-
ror, r2_score
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_er-
ror
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
#Построим график распределения целевой переменной (popularity)
plt.figure(figsize=(10, 6))
sns.histplot(data[target], kde=True, bins=30)
plt.title('Распределение целевой переменной (popularity)')
plt.xlabel('Popularity Score')
plt.ylabel('Частота')
plt.show()
Линейная регрессия без кросс-валидации
# Выбор признаков и целевой переменной
X = data[features]  # Признаки (независимые переменные)
y = data[target]    # Целевая переменная (зависимая переменная)

# Разделение данных на обучающую и тестовую выборки (80% - обуча-
ющая, 20% - тестовая)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=50)
# Создание экземпляра StandardScaler для нормализации признаков
scaler = StandardScaler()
# Нормализация признаков: вычисляется среднее и стандартное от-
клонение на обучающей выборке
```

```
X_train_scaled = scaler.fit_transform(X_train)
# Применение нормализации к тестовой выборке на основе параметров обучающей выборки
X_test_scaled = scaler.transform(X_test)
# Создание и обучение модели линейной регрессии
model = LinearRegression()
model.fit(X_train_scaled, y_train)
# Предсказание целевой переменной на тестовой выборке
y_pred = model.predict(X_test_scaled)
# Оценка модели с использованием различных метрик
mse = mean_squared_error(y_test, y_pred)          # Среднеквадратичная ошибка
mae = mean_absolute_error(y_test, y_pred)         # Средняя абсолютная ошибка
rmse = np.sqrt(mse)                                # Корень из среднеквадратичной ошибки
# Составим график распределения ошибок (остатков) для проверки нормальности распределения ошибок
residuals = y_test - y_pred
plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True, bins=30, color='blue')
plt.axvline(0, color='r', linestyle='--', linewidth=2)
plt.title('Распределение ошибок (Linear Regression)')
plt.xlabel('Ошибка предсказания')
plt.ylabel('Частота')
plt.grid(True, linestyle='--', alpha=0.3)
plt.show()
#Составим график фактических vs предсказанных значений
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.3)
plt.plot([y.min(), y.max()], [y.min(), y.max()], '--r')
plt.xlabel('Фактические значения')
plt.ylabel('Предсказанные значения')
plt.title(f'Фактические vs Предсказанные ({model.__class__.__name__})')
```

```
plt.grid(True)
plt.show()
# Линейная регрессия с кросс-валидацией
# Выбор признаков и целевой переменной
X = data[features]  # Признаки (независимые переменные)
y = data[target]    # Целевая переменная (зависимая переменная)
# Разделение данных на обучающую и тестовую выборки (80% - обуча-
ющая, 20% - тестовая)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=50)
# Создание экземпляра StandardScaler для нормализации признаков
scaler = StandardScaler()
# Нормализация признаков: вычисляется среднее и стандартное от-
клонение на обучающей выборке
X_train_scaled = scaler.fit_transform(X_train)
# Применение нормализации к тестовой выборке
X_test_scaled = scaler.transform(X_test)
# Создание модели линейной регрессии
model = LinearRegression()
# Кросс-валидация с 5 фолдами для оценки метрик (MSE, MAE, R^2)
cv_mse = cross_val_score(model, X_train_scaled, y_train, cv=5,
scoring='neg_mean_squared_error')
cv_mae = cross_val_score(model, X_train_scaled, y_train, cv=5,
scoring='neg_mean_absolute_error')
cv_r2 = cross_val_score(model, X_train_scaled, y_train, cv=5,
scoring='r2')
# Преобразование отрицательных значений для MSE и MAE в положи-
тельные
cv_mse = -cv_mse
cv_mae = -cv_mae
# Вывод результатов кросс-валидации
print('Результаты:')
print(f'Mean Squared Error (MSE): {np.mean(cv_mse):.2f} ±
{np.std(cv_mse):.2f}')  # Среднеквадратичная ошибка
```

```
print(f'Mean Absolute Error (MAE): {np.mean(cv_mae):.2f} ± {np.std(cv_mae):.2f}') # Средняя абсолютная ошибка
print(f'R^2 Score: {np.mean(cv_r2):.2f} ± {np.std(cv_r2):.2f}') # Коэффициент детерминации
# Составим график фактических vs предсказанных значений
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.3)
plt.plot([y.min(), y.max()], [y.min(), y.max()], '--r')
plt.xlabel('Фактические значения')
plt.ylabel('Предсказанные значения')
plt.title(f'Фактические vs Предсказанные ({model.__class__.__name__})')
plt.grid(True)
plt.show()
#Сравнение моделей линейной регрессии
# Средние значения метрик для кросс-валидации
cv_mse_mean = np.mean(cv_mse)  # Среднее значение среднеквадратичной ошибки (MSE) из кросс-валидации
cv_r2_mean = np.mean(cv_r2)    # Среднее значение коэффициента детерминации (R²) из кросс-валидации
# Подготовка данных для графиков
metrics_simple = {'MSE': mse, 'R2': r2}  # Метрики без кросс-валидации
metrics_cv = {'MSE': cv_mse_mean, 'R2': cv_r2_mean}  # Метрики с кросс-валидацией
# Построение графиков
fig, ax = plt.subplots(1, 2, figsize=(15, 5))  # Создание полотна с 2 подграфиками

# График для MSE
ax[0].bar(['Без кросс-валидации', 'С кросс-валидацией'],
          [metrics_simple['MSE'], metrics_cv['MSE']],
color=['green', 'purple'])  # Столбчатая диаграмма
ax[0].set_title('Среднеквадратичная ошибка (MSE)')  # Заголовок графика
```

```
ax[0].set_ylabel('MSE')  # Подпись оси Y

# График для R²
ax[1].bar(['Без кросс-валидации', 'С кросс-валидацией'],
          [metrics_simple['R2'],              metrics_cv['R2']],
color=['green', 'purple'])  # Столбчатая диаграмма
ax[1].set_title('Коэффициент детерминации (R²)')  # Заголовок гра-
фика
ax[1].set_ylabel('R²')  # Подпись оси Y

# Общий заголовок для всего полотна
plt.suptitle('Сравнение моделей линейной регрессии')
plt.tight_layout()  # Автоматическая настройка отступов между гра-
фиками
plt.show()  # Отображение графиков
#Градиентный бустинг без кросс-валидации
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_er-
ror, r2_score
# Выбор признаков и целевой переменной
X = data[features]  # Признаки (независимые переменные)
y = data[target]    # Целевая переменная (зависимая переменная)

# Разделение данных на обучающую и тестовую выборки (80% - обуча-
ющая, 20% - тестовая)
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,
test_size=0.2, random_state=50)

# Нормализация признаков
#scaler = StandardScaler()  # Стандартный скейлер (включен для
будущего использования)
#X_train_scaled = scaler.fit_transform(X_train)  # Нормализация
обучающей выборки
```

```
#X_test_scaled = scaler.transform(X_test)  # Нормализация тестовой выборки
X_train_scaled = X_train  # Используем необработанные данные для обучения
X_test_scaled = X_test    # Используем необработанные данные для тестирования
# Создание и обучение модели градиентного бустинга
model = GradientBoostingRegressor(
    n_estimators=50,           # Количество деревьев в ансамбле
    learning_rate=0.2,          # Темп обучения
    max_depth=3,                # Максимальная глубина деревьев
    random_state=50            # Установление случайного состояния для воспроизводимости
)
model.fit(X_train_scaled, y_train)  # Обучение модели на обучающей выборке

# Предсказание целевой переменной на тестовой выборке
y_pred = model.predict(X_test_scaled)  # Предсказание для тестовой выборки

# Оценка модели
mse = mean_squared_error(y_test, y_pred)           # Вычисление среднеквадратичной ошибки
mae = mean_absolute_error(y_test, y_pred)          # Вычисление средней абсолютной ошибки
r2 = r2_score(y_test, y_pred)                       # Вычисление коэффициента детерминации
# Вывод результатов
print('Результаты оценки модели:')
print(f'Mean Squared Error (MSE): {mse:.2f}')     # Вывод MSE
print(f'Mean Absolute Error (MAE): {mae:.2f}')    # Вывод MAE
print(f'R^2 Score: {r2:.2f}')                     # Вывод R²
#Строим график распределения ошибок
```

```
residuals = y_test - y_pred

plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True, bins=30, color='green')
plt.axvline(0, color='r', linestyle='--', linewidth=2)
plt.title('Распределение ошибок (Gradient Boosting)')
plt.xlabel('Ошибка предсказания')
plt.ylabel('Частота')
plt.grid(True, linestyle='--', alpha=0.3)
plt.show()
# Строим график фактических и предсказанных значений

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.3)
plt.plot([y.min(), y.max()], [y.min(), y.max()], '--r')
plt.xlabel('Фактические значения')
plt.ylabel('Предсказанные значения')
plt.title(f'Фактические vs Предсказанные ({model.__class__.__name__})')
plt.grid(True)
plt.show()
#Благодаря интерпретируемости этой модели можно посмотреть важность признаков, построим соответствующий график
# Получаем важность признаков из обученной модели градиентного бустинга
feature_importances = model.feature_importances_  # Важность каждого признака
feature_names = X_train.columns  # Названия признаков
# Сортируем признаки по важности в порядке убывания
sorted_index = np.argsort(feature_importances)[::-1]  # Индексы, отсортированные по важности
sorted_feature_importences = feature_importances[sorted_index]  # Важности, отсортированные
sorted_feature_names = feature_names[sorted_index]  # Названия признаков в порядке убывания важности
```

```
# Выбор топ-25 наиболее важных признаков
top_25_feature_importances = sorted_feature_importences[:25]   # Важности топ-25 признаков
top_25_feature_names = sorted_feature_names[:25]  # Названия топ-25 признаков
# Строим гистограмму для первых 50 наиболее важных признаков
plt.figure(figsize=(10, 6))  # Настройка размера графика
plt.bar(range(len(top_25_feature_importances)),       top_25_feature_importances, color='red')  # Создание столбчатой диаграммы
plt.xticks(range(len(top_25_feature_importances)),    top_25_feature_names, rotation=90)  # Настройка меток по оси X
plt.title('Важность признаков для градиентного бустинга (топ 25)')
# Заголовок графика
plt.xlabel("Признаки")  # Подпись оси X
plt.ylabel("Важность")  # Подпись оси Y
plt.tight_layout()  # Автоматическая настройка отступов
plt.show()  # Отображение графика
#Градиентный бустинг с кросс-валидацией
Градиентный бустинг с кросс-валидацией - это метод, который:

from sklearn.ensemble import GradientBoostingRegressor
from     sklearn.model_selection     import     train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
# Выбор признаков и целевой переменной
X = data[features]  # Признаки (независимые переменные)
y = data[target]    # Целевая переменная (зависимая переменная)

# Разделение данных на обучающую и тестовую выборки (80% - обучающая, 20% - тестовая)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=50)

# Создание экземпляра StandardScaler для нормализации признаков
scaler = StandardScaler()  # Инициализация стандартного скейлера

# Нормализация признаков: вычисляется среднее и стандартное от-
клонение на обучающей выборке
X_train_scaled = scaler.fit_transform(X_train)   # Нормализация
обучающей выборки

# Применение нормализации к тестовой выборке
X_test_scaled = scaler.transform(X_test)  # Нормализация тестовой
выборки с учетом параметров обучающей выборки
# Создание модели градиентного бустинга
model = GradientBoostingRegressor(
    n_estimators=50,  # Количество деревьев в ансамбле
    learning_rate=0.2,  # Темп обучения
    max_depth=3,        # Максимальная глубина каждого дерева
    random_state=50      # Установление случайного состояния для
воспроизводимости
)

# Кросс-валидация с 5 фолдами для оценки метрик (MSE, MAE, R²)
cv_mse = cross_val_score(model, X_train_scaled, y_train, cv=5,
scoring='neg_mean_squared_error')  # Кросс-валидация для MSE
cv_mae = cross_val_score(model, X_train_scaled, y_train, cv=5,
scoring='neg_mean_absolute_error')  # Кросс-валидация для MAE
cv_r2 = cross_val_score(model, X_train_scaled, y_train, cv=5,
scoring='r2')  # Кросс-валидация для R²

# Преобразование отрицательных значений для MSE и MAE в положи-
тельные
cv_mse = -cv_mse  # Преобразуем в положительные значения
cv_mae = -cv_mae  # Преобразуем в положительные значения
```

```
# Вывод результатов кросс-валидации
print('Результаты кросс-валидации (5 фолдов):')
print(f'Mean Squared Error (MSE): {np.mean(cv_mse):.2f} ±
{np.std(cv_mse):.2f}') # Среднее и стандартное отклонение MSE
print(f'Mean Absolute Error (MAE): {np.mean(cv_mae):.2f} ±
{np.std(cv_mae):.2f}') # Среднее и стандартное отклонение MAE
print(f'R^2 Score: {np.mean(cv_r2):.2f} ± {np.std(cv_r2):.2f}') #
Среднее и стандартное отклонение R²

#Строим график распределения ошибок

residuals = y_test - y_pred

plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True, bins=30, color='green')
plt.axvline(0, color='r', linestyle='--', linewidth=2)
plt.title('Распределение ошибок (Gradient Boosting)')
plt.xlabel('Ошибка предсказания')
plt.ylabel('Частота')
plt.grid(True, linestyle='--', alpha=0.3)
plt.show()
#Строим график фактических и предсказанных значений
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.3)
plt.plot([y.min(), y.max()], [y.min(), y.max()], '--r')
plt.xlabel('Фактические значения')
plt.ylabel('Предсказанные значения')
plt.title(f'Фактические vs Предсказанные
({model.__class__.__name__})')
plt.grid(True)
plt.show()
#Сравнение моделей градиентного бустинга
# Средние значения метрик для кросс-валидации
```

```
cv_mse_mean = np.mean(cv_mse)  # Среднее значение среднеквадратичной ошибки (MSE) для кросс-валидации
cv_r2_mean = np.mean(cv_r2)    # Среднее значение коэффициента детерминации (R²) для кросс-валидации

# Данные для графиков
metrics_simple = {'MSE': mse, 'R2': r2}  # Метрики без кросс-валидации
metrics_cv = {'MSE': cv_mse_mean, 'R2': cv_r2_mean}  # Метрики с кросс-валидацией

# Построение графиков
fig, ax = plt.subplots(1, 2, figsize=(15, 5))  # Создаем полотно с двумя подграфиками

# График для MSE
ax[0].bar(['Без кросс-валидации', 'С кросс-валидацией'],
          [metrics_simple['MSE'],            metrics_cv['MSE']],
color=['green', 'purple'])  # Столбчатая диаграмма для MSE
ax[0].set_title('Среднеквадратичная ошибка (MSE)')  # Заголовок для MSE
ax[0].set_ylabel('MSE')  # Подпись оси Y для MSE

# График для R²
ax[1].bar(['Без кросс-валидации', 'С кросс-валидацией'],
          [metrics_simple['R2'],              metrics_cv['R2']],
color=['green', 'purple'])  # Столбчатая диаграмма для R²
ax[1].set_title('Коэффициент детерминации (R²)')  # Заголовок для R²
ax[1].set_ylabel('R²')  # Подпись оси Y для R²

# Общий заголовок для всего полотна
plt.suptitle('Сравнение моделей градиентного бустинга')  # Заголовок для всей визуализации
```

```
plt.tight_layout()  # Оптимизация отступов для лучшего отображения графиков
plt.show()  # Отображение графиков
#Сравнение моделей по среднеквадратичной ошибке MSE и коэффициенту детерминации R²
metrics = {
    'Linear Regression': {
        'MSE': mse,                 # Из блока линейной регрессии
        'R2': r2                    # Из блока линейной регрессии
    },
    'Gradient Boosting': {
        'MSE': mse,                      # Из блока градиентного бустинга
        'R2': r2                         # Из блока градиентного бустинга
    }
}

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

# График MSE
pd.DataFrame.from_dict(metrics, orient='index')['MSE'].plot(
    kind='bar', ax=ax1, color=['skyblue', 'salmon'])
ax1.set_title('Сравнение MSE моделей')
ax1.set_ylabel('MSE')
ax1.grid(True, linestyle='--', alpha=0.6)

# График R2
pd.DataFrame.from_dict(metrics, orient='index')['R2'].plot(
    kind='bar', ax=ax2, color=['skyblue', 'salmon'])
ax2.set_title('Сравнение R² моделей')
ax2.set_ylabel('R²')
ax2.grid(True, linestyle='--', alpha=0.6)

plt.tight_layout()
```

```
plt.show()
#Сравнение моделей по важности признаков
try:
    # Получаем коэффициенты из LinearRegression (из вашего первого блока)
    lr_model = LinearRegression()
    lr_model.fit(X_train_scaled, y_train)  # используем ваши переменные
    lr_coefficients = lr_model.coef_

    # Получаем важность признаков из GradientBoosting (из вашего второго блока)
    gb_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
    gb_model.fit(X_train, y_train)  # GB у вас работает без масштабирования
    gb_importances = gb_model.feature_importances_

    # Создаем DataFrame для визуализации
    importance_df = pd.DataFrame({
        'Feature': features,
        'LinearRegression': np.abs(lr_coefficients),  # берем модуль коэффициентов
        'GradientBoosting': gb_importances
    })

    # Нормализуем значения от 0 до 1
    importance_df['LinearRegression'] = importance_df['LinearRegression'] / importance_df['LinearRegression'].max()
    importance_df['GradientBoosting'] = importance_df['GradientBoosting'] / importance_df['GradientBoosting'].max()

    # Сортируем по важности в GB
    importance_df = importance_df.sort_values('GradientBoosting', ascending=False).head(15)
```

```
    # Строим график
    plt.figure(figsize=(12, 8))

    # График для LinearRegression
    plt.subplot(1, 2, 1)
    sns.barplot(x='LinearRegression', y='Feature', data=im-
portance_df, palette='Blues_d')
    plt.title('Топ-15 признаков (LinearRegression)\nНормированные
коэффициенты')

    # График для GradientBoosting
    plt.subplot(1, 2, 2)
    sns.barplot(x='GradientBoosting', y='Feature', data=im-
portance_df, palette='Oranges_d')
    plt.title('Топ-15 признаков (GradientBoosting)\nНормированная
важность')

    plt.tight_layout()
    plt.show()

except Exception as e:
    print(f"Ошибка: {e}\nУбедитесь, что:")
    print("1. Вы выполнили все предыдущие блоки кода")
    print("2. Переменные features, X_train, y_train существуют")
```