Geekbrains

**Разработка системы генерации и визуализации CSV файлов для оптимизации аналитических процессов, повышения эффективности работы с данными, поддержки обучения и тестирования моделей машинного обучения**

**Программа:**

Разработчик

**Специализация:**

Аналитик

Программист

Data Engineer

Fullstack разработчик

ФИО

Вересова Анастасия Владимировна

Санкт-Петербург

2025

**Содержание**

# Введение

# 1.1. Актуальность темы

# Описание актуальности темы дипломного проекта.

# Объяснение, почему важно иметь инструменты для генерации и анализа данных.

# 1.2. Цели и задачи работы

# Основные цели проекта.

# Конкретные задачи, которые необходимо решить для достижения целей.

# 1.3. Обзор существующих решений

# Краткий обзор существующих инструментов и приложений для генерации и анализа данных.

# Выделение их недостатков и преимуществ.

# Теоретическая часть

# 2.1. Основные концепции и технологии

# Описание основных концепций, связанных с обработкой и анализом данных (CSV, JSON, машинное обучение и т.д.).

# Обзор библиотек и инструментов, используемых в проекте (PapaParse, TensorFlow.js, Chart.js и т.д.).

# 2.2. Форматы данных

# Описание форматов данных, используемых в проекте (CSV, JSON, Excel).

# Преимущества и недостатки каждого формата.

# 2.3. Методы анализа данных

# Описание методов анализа данных, используемых в проекте (анализ пропусков, уникальных и повторяющихся значений, визуализация данных).

# Примеры применения этих методов.

# 2.4. Машинное обучение

# Основные концепции машинного обучения.

# Описание библиотеки TensorFlow.js и ее возможностей.

# Примеры создания и обучения моделей машинного обучения.

# Практическая часть

# 3.1. Описание разработанного приложения

# Общее описание веб-приложения.

# Основные функции и возможности приложения.

# 3.2. Архитектура приложения

# Описание архитектуры приложения.

# Взаимодействие между различными компонентами приложения.

# 3.3. Интерфейс пользователя

# Описание интерфейса пользователя.

# Основные элементы интерфейса и их назначение.

# 3.4. Генерация данных

# Подробное описание функции генерации данных.

# Примеры использования различных типов данных и диапазонов.

# 3.4.2 Валидация данных

# Подробное описание функции генерации данных.

# 3.5. Загрузка и сохранение данных

# Описание функции загрузки CSV файлов.

# Описание функции сохранения данных в различных форматах (CSV, Excel, JSON).

# 3.6. Анализ данных

# Подробное описание функции анализа данных.

# Примеры анализа данных на наличие пропусков, уникальных и повторяющихся значений.

# Визуализация данных с помощью графиков и тепловых карт.

# 3.7. Интеграция с машинным обучением

# Описание функции создания и обучения моделей машинного обучения.

# Примеры использования моделей машинного обучения для анализа данных.

# 3.8. Примеры использования приложения

# Примеры использования приложения для различных задач.

# Пошаговые инструкции по использованию приложения.

# Результаты и обсуждение

# 4.1. Анализ результатов

# Анализ результатов работы приложения.

# Обсуждение преимуществ и недостатков разработанного решения.

# 4.2. Сравнение с существующими решениями

# Сравнение разработанного приложения с существующими инструментами и приложениями.

# Выделение уникальных особенностей и преимуществ вашего приложения.

# 4.3. Возможные улучшения

# Обсуждение возможных улучшений и дополнений к приложению.

# Рекомендации по дальнейшему развитию проекта.

# Заключение

# 5.1. Выводы

# Основные выводы по результатам работы.

# Достижение поставленных целей и задач.

# 5.2. Рекомендации

# Рекомендации по использованию разработанного приложения.

# Рекомендации по дальнейшему развитию проекта.

# Приложения

# 6.1. Исходный код приложения

# Полный исходный код приложения.

# Комментарии и пояснения к коду.

# 6.2. Документация и инструкции по использованию

# Пользовательская документация.

# Инструкции по установке и запуску приложения.

# 6.3. Примеры данных. Скриншоты и визуализации

# Примеры данных, используемых в проекте.

# Примеры сгенерированных данных.

# Актуальность темы

# Введение

# В современном мире данные играют ключевую роль в принятии решений, разработке стратегий и оптимизации процессов во всех сферах деятельности. От бизнеса и науки до здравоохранения и образования — данные становятся основой для инноваций и прогресса. В этом контексте инструменты для генерации и анализа данных приобретают особую значимость.

# Почему важно иметь инструменты для генерации и анализа данных

# 1. Ускорение процесса принятия решений

# Современные организации сталкиваются с огромным объемом данных, которые необходимо анализировать для принятия обоснованных решений. Инструменты для генерации и анализа данных позволяют автоматизировать этот процесс, значительно сокращая время, необходимое для анализа и интерпретации данных. Это особенно важно в условиях быстро меняющейся рыночной среды, где оперативность решений может стать ключевым фактором успеха.

# 2. Повышение точности и надежности данных

# Генерация данных с заданными параметрами и диапазонами позволяет создавать тестовые наборы данных, которые могут быть использованы для проверки гипотез, тестирования алгоритмов и моделей машинного обучения. Это обеспечивает высокую точность и надежность данных, что особенно важно в научных исследованиях и разработке новых технологий.

# 3. Оптимизация бизнес-процессов

# Анализ данных позволяет выявлять скрытые закономерности, тренды и аномалии, что может быть использовано для оптимизации бизнес-процессов. Например, анализ данных о продажах может помочь выявить наиболее прибыльные продукты или сегменты рынка, что позволит компании сфокусировать свои усилия на этих направлениях.

# 4. Улучшение качества обслуживания клиентов

# Анализ данных о взаимодействии с клиентами позволяет выявлять проблемы и потребности клиентов, что помогает улучшить качество обслуживания и повысить уровень удовлетворенности клиентов. Это особенно важно в условиях высокой конкуренции, где качество обслуживания может стать решающим фактором при выборе поставщика услуг.

# 5. Развитие научных исследований

# Инструменты для генерации и анализа данных играют ключевую роль в научных исследованиях, где необходимо обрабатывать и анализировать большие объемы данных. Это позволяет ученым выявлять новые закономерности, проверять гипотезы и делать открытия, которые могут иметь значительное влияние на развитие науки и технологий.

# 6. Образование и обучение

# Инструменты для генерации и анализа данных могут быть использованы в образовательных учреждениях для обучения студентов методам анализа данных, статистике и машинному обучению. Это помогает подготовить специалистов, которые будут востребованы на рынке труда и смогут внести значительный вклад в развитие различных отраслей.

# 7. Здравоохранение

# В здравоохранении анализ данных позволяет выявлять закономерности в заболеваемости, эффективности лечения и других аспектах, что помогает улучшить качество медицинских услуг и разрабатывать новые методы лечения. Это особенно важно в условиях пандемии, когда оперативный анализ данных может спасти жизни.

# Заключение

# Таким образом, инструменты для генерации и анализа данных являются неотъемлемой частью современного мира, где данные играют ключевую роль в принятии решений, разработке стратегий и оптимизации процессов. Разработка эффективных и удобных инструментов для работы с данными позволяет значительно повысить точность, надежность и оперативность анализа, что имеет важное значение для бизнеса, науки, здравоохранения и образования.

# 1.2. Цели и задачи работы

# Основные цели проекта

# Разработка веб-приложения для генерации и анализа CSV данных:

# Создание удобного и функционального инструмента, который позволит пользователям генерировать тестовые наборы данных с заданными параметрами и диапазонами с осмысленным заполнением.

# Интеграция функций загрузки и сохранения данных:

# Обеспечение возможности загрузки CSV файлов для анализа и сохранения сгенерированных данных в различных форматах (CSV, Excel, JSON).

# Интеграция с машинным обучением:

# Создание и обучение моделей машинного обучения на основе загруженных данных с использованием библиотеки TensorFlow.js.

# Обеспечение возможности прогнозирования и визуализации результатов модели.

# Обеспечение удобного интерфейса пользователя:

# Разработка интуитивно понятного интерфейса, который позволит пользователям легко взаимодействовать с приложением и использовать его функции.

# Конкретные задачи, которые необходимо решить для достижения целей

# Обеспечение отзывчивости и адаптивности интерфейса для различных устройств и экранов.

# Разработка функции генерации данных:

# Реализация алгоритмов для генерации данных различных типов (даты, время, числа, строки и т.д.) с заданными диапазонами и параметрами.

# Обеспечение возможности добавления и удаления столбцов, а также выбора типов данных и диапазонов.

# Реализация функционала для создания больших данных.

# Реализация функции загрузки CSV файлов:

# Разработка механизма для загрузки CSV файлов и их анализа.

# Обработка и очистка данных, включая удаление пустых строк и замена пропущенных значений.

# Реализация функционала для загрузки больших данных.

# Создание функции сохранения данных:

# Реализация возможности сохранения сгенерированных данных в форматах CSV, Excel и JSON.

# Обеспечение корректного форматирования данных при сохранении.

# Реализация функционала для сохранения больших данных.

# Разработка функций анализа данных:

# Внедрение методов анализа данных, включая выявление пропусков, уникальных и повторяющихся значений, а также визуализацию данных с помощью графиков и тепловых карт.

# Тестирование и оптимизация приложения:

# Проведение тщательного тестирования всех функций приложения для обеспечения их корректной работы.

# Оптимизация производительности приложения, особенно при работе с большими объемами данных.

# 1.3. Обзор существующих решений

# Краткий обзор существующих инструментов и приложений для генерации и анализа данных

# Microsoft Excel:

# Преимущества: Широко используемый инструмент для работы с табличными данными, поддержка различных форматов, встроенные функции анализа данных.

# Недостатки: Ограниченные возможности для автоматической генерации данных, сложность работы с большими объемами данных, отсутствие интеграции с машинным обучением.

# Google Sheets:

# Преимущества: Облачное решение, поддержка совместной работы, интеграция с другими сервисами Google.

# Недостатки: Ограниченные возможности для автоматической генерации данных, отсутствие интеграции с машинным обучением.

# Pandas (Python):

# Преимущества: Мощная библиотека для работы с данными, поддержка различных форматов, возможность интеграции с машинным обучением.

# Недостатки: Требует знания программирования на Python, сложность использования для нетехнических пользователей.

# Tableau:

# Преимущества: Мощный инструмент для визуализации данных, поддержка различных источников данных, возможность создания интерактивных дашбордов.

# Недостатки: Высокая стоимость лицензии, сложность использования для нетехнических пользователей, ограниченные возможности для автоматической генерации данных.

# Power BI:

# Преимущества: Интеграция с различными источниками данных, возможность создания интерактивных отчетов и дашбордов, поддержка машинного обучения.

# Недостатки: Сложность использования для нетехнических пользователей, ограниченные возможности для автоматической генерации данных.

# Выделение их недостатков и преимуществ

# Существующие инструменты и приложения для генерации и анализа данных имеют свои преимущества и недостатки. Microsoft Excel и Google Sheets являются популярными инструментами для работы с табличными данными, но они имеют ограниченные возможности для автоматической генерации данных и интеграции с машинным обучением. Pandas (Python) предоставляет мощные возможности для работы с данными, но требует знания программирования. Tableau и Power BI являются мощными инструментами для визуализации данных, но они могут быть сложными в использовании для нетехнических пользователей и имеют ограниченные возможности для автоматической генерации данных.

**Теоретическая часть**

**2.1. Основные концепции и технологии**

**Описание основных концепций, связанных с обработкой и анализом данных**

**CSV (Comma-Separated Values)**

CSV — это текстовый формат, используемый для хранения табличных данных. В CSV данные разделяются запятыми, что делает его удобным для чтения и записи. Этот формат широко используется для обмена данными между различными системами и приложениями.

**JSON (JavaScript Object Notation)**

JSON — это легковесный формат обмена данными, который легко читается и пишется человеком, а также легко парсится и генерируется машинами. JSON часто используется для передачи данных между сервером и клиентом в веб-приложениях.

**Машинное обучение**

Машинное обучение — это область искусственного интеллекта, которая занимается разработкой алгоритмов и моделей, позволяющих компьютерам обучаться на данных и делать прогнозы или принять решения без явного программирования.

**Обзор библиотек и инструментов, используемых в проекте**

**PapaParse**

PapaParse — это быстрая и мощная библиотека для парсинга CSV файлов в JavaScript. Она поддерживает различные форматы CSV и предоставляет удобные методы для работы с данными.

**TensorFlow.js**

TensorFlow.js — это библиотека для машинного обучения, которая позволяет разрабатывать и обучать модели машинного обучения непосредственно в браузере или на сервере с использованием JavaScript. Она предоставляет широкий набор инструментов для создания, обучения и развертывания моделей машинного обучения.

**Chart.js**

Chart.js — это простая, но мощная библиотека для создания графиков и диаграмм в JavaScript. Она поддерживает различные типы графиков, такие как линейные, столбчатые, круговые и другие, и предоставляет удобные методы для визуализации данных.

**Простота использования:**

Chart.js предоставляет простой и интуитивно понятный API для создания графиков.

Поддерживает различные типы графиков, такие как линейные, столбчатые, круговые, точечные, радиальные и другие.

**Настраиваемость:**

Библиотека позволяет легко настраивать внешний вид графиков, включая цвета, шрифты, размеры и другие параметры.

Поддерживает анимации для плавного отображения данных.

**Интерактивность:**

Графики могут быть интерактивными, с поддержкой событий, таких как наведение курсора и клики.

Поддерживает легенды и подсказки для отображения дополнительной информации.

**Производительность:**

Chart.js оптимизирована для работы с большими объемами данных и обеспечивает высокую производительность.

**Поддержка плагинов:**

Библиотека поддерживает плагины, которые позволяют расширять функциональность графиков.

**Plotly**

**Plotly** — это мощная библиотека для создания интерактивных графиков и диаграмм. Она поддерживает широкий спектр типов графиков и предоставляет богатые возможности для визуализации данных.

**Широкий спектр графиков:**

Plotly поддерживает более 40 типов графиков, включая линейные, столбчатые, круговые, точечные, 3D-графики, тепловые карты и многое другое.

**Интерактивность:**

Графики Plotly являются высоко интерактивными, с поддержкой зума, панорамирования, наведения курсора и кликов.

Поддерживает легенды и подсказки для отображения дополнительной информации.

**Настраиваемость:**

Библиотека предоставляет широкие возможности для настройки внешнего вида графиков, включая цвета, шрифты, размеры и другие параметры.

Поддерживает анимации для плавного отображения данных.

**Интеграция с другими инструментами:**

Plotly легко интегрируется с другими инструментами и библиотеками, такими как Dash для создания веб-приложений.

**Поддержка больших данных:**

Plotly оптимизирована для работы с большими объемами данных и обеспечивает высокую производительность.

**XLSX**

XLSX — это библиотека для работы с Excel файлами в JavaScript. Она позволяет читать и записывать данные в формате Excel, что делает ее удобной для экспорта данных в этот формат.

**2.2. Форматы данных**

**Описание форматов данных, используемых в проекте**

**CSV**

CSV — это текстовый формат, который используется для хранения табличных данных. Данные в CSV файле разделяются запятыми, что делает его удобным для чтения и записи. CSV широко используется для обмена данными между различными системами и приложениями.

**JSON**

JSON — это легковесный формат обмена данными, который легко читается и пишется человеком, а также легко парсится и генерируется машинами. JSON часто используется для передачи данных между сервером и клиентом в веб-приложениях.

**Excel**

Excel — это формат электронных таблиц, который используется для хранения и анализа данных. Excel поддерживает различные функции для работы с данными, такие как формулы, графики и диаграммы.

**Преимущества и недостатки каждого формата**

**CSV**

Преимущества: Простота, легкость чтения и записи, широкая поддержка различными системами и приложениями.

Недостатки: Ограниченные возможности для форматирования данных, отсутствие поддержки сложных структур данных.

**JSON**

Преимущества: Легковесность, удобство чтения и записи, поддержка сложных структур данных, широкое использование в веб-приложениях.

Недостатки: Меньшая читаемость для человека по сравнению с CSV, отсутствие поддержки формул и графиков.

**Excel**

Преимущества: Поддержка формул, графиков и диаграмм, удобство работы с данными, широкая поддержка различными системами и приложениями.

Недостатки: Большой размер файлов, сложность работы с большими объемами данных, необходимость специального программного обеспечения для работы с файлами.

**2.3. Методы анализа данных**

**Описание методов анализа данных, используемых в проекте**

**Анализ пропусков**

Анализ пропусков позволяет выявить отсутствующие или пустые значения в данных. Это важно для очистки данных и обеспечения их корректного анализа.

**Анализ уникальных и повторяющихся значений**

Анализ уникальных и повторяющихся значений позволяет выявить уникальные значения в данных, а также определить, какие значения повторяются. Это помогает понять структуру данных и выявить возможные аномалии.

**Визуализация данных**

Визуализация данных позволяет представлять данные в виде графиков и диаграмм, что помогает лучше понять структуру данных и выявить закономерности. Визуализация данных является важным инструментом для анализа данных и принятия решений.

**Примеры применения этих методов**

**Анализ пропусков**

Пример: Анализ данных о продажах может выявить пропуски в данных о количестве проданных товаров, что поможет очистить данные и обеспечить их корректный анализ.

**Анализ уникальных и повторяющихся значений**

Пример: Анализ данных о клиентах может выявить уникальные и повторяющиеся значения в данных о контактной информации, что поможет определить, какие клиенты являются уникальными, а какие — повторяющимися.

**Визуализация данных**

Пример: Визуализация данных о продажах в виде графиков может помочь выявить тренды и закономерности в продажах, что поможет принять обоснованные решения по улучшению продаж.

**2.4. Машинное обучение**

**Основные концепции машинного обучения**

**Машинное обучение** — это область искусственного интеллекта, которая занимается разработкой алгоритмов и моделей, позволяющих компьютерам обучаться на данных и делать прогнозы или принять решения без явного программирования. Основные концепции машинного обучения включают:

**Обучающие данные:** Данные, используемые для обучения модели.

**Обучающие алгоритмы:** Алгоритмы, используемые для обучения модели на данных.

**Модели машинного обучения:** Модели, которые обучаются на данных и используются для прогнозирования или принятия решений.

**Описание библиотеки TensorFlow.js и ее возможностей**

**TensorFlow.js** — это библиотека для машинного обучения, которая позволяет разрабатывать и обучать модели машинного обучения непосредственно в браузере или на сервере с использованием JavaScript. TensorFlow.js предоставляет широкий набор инструментов для создания, обучения и развертывания моделей машинного обучения, включая:

**Создание моделей:** Инструменты для создания моделей машинного обучения.

**Обучение моделей:** Алгоритмы для обучения моделей на данных.

**Прогнозирование:** Методы для использования обученных моделей для прогнозирования.

**Визуализация:** Инструменты для визуализации данных и результатов моделей.

**Примеры создания и обучения моделей машинного обучения**

**Пример создания модели**

const model = tf.sequential();

model.add(tf.layers.dense({ units: 128, activation: 'relu', inputShape: [inputShape] }));

model.add(tf.layers.dense({ units: yTrain.shape[1], activation: 'linear' }));

**Пример обучения модели**

model.compile({

optimizer: 'adam',

loss: 'meanSquaredError',

metrics: ['mae']

});

await model.fit(xTrain, yTrain, {

epochs: 10,

batchSize: 32,

callbacks: {

onEpochEnd: (epoch, logs) => console.log(`Epoch ${epoch + 1}: loss = ${logs.loss}`)

}

});

**Пример прогнозирования**

const predictions = model.predict(xTest);

console.log('Прогнозы:', predictions);

**Практическая часть**

**3.1. Описание разработанного приложения**

**Общее описание веб-приложения**

CSV Generator — это веб-приложение, предназначенное для генерации, анализа и управления CSV данными. Оно позволяет пользователям создавать тестовые наборы данных с заданными параметрами и диапазонами, загружать и сохранять CSV файлы, анализировать данные и визуализировать результаты. Приложение также поддерживает интеграцию с машинным обучением, что позволяет создавать и обучать модели на основе загруженных данных.

**Основные функции и возможности приложения**

**Генерация данных:**

Создание данных различных типов (даты, время, числа, строки и т.д.) с заданными диапазонами и параметрами.

Возможность добавления и удаления столбцов, а также выбора типов данных и диапазонов.

**Загрузка и сохранение данных:**

Поддержка загрузки CSV файлов для анализа.

Возможность сохранения сгенерированных данных в форматах CSV, Excel и JSON.

**Анализ данных:**

Анализ данных на наличие пропусков, уникальных и повторяющихся значений.

Визуализация данных с помощью графиков и тепловых карт.

Интеграция с машинным обучением:

Возможность создания и обучения моделей машинного обучения на основе загруженных данных.

Прогнозирование и визуализация результатов модели.

**3.2. Архитектура приложения**

**Описание архитектуры приложения**

Приложение построено на основе веб-технологий и включает в себя следующие основные компоненты:

**Интерфейс пользователя (UI):**

HTML и CSS для создания структуры и стилей интерфейса.

JavaScript для реализации логики взаимодействия с пользователем.

Пользователь взаимодействует с приложением через веб-интерфейс, который предоставляет формы для ввода параметров, кнопки для выполнения действий и модальные окна для отображения результатов.

**Библиотеки и инструменты:**

PapaParse для парсинга CSV файлов.

TensorFlow.js для машинного обучения.

Chart.js и Plotly для визуализации данных.

XLSX для работы с Excel файлами.

**Хранилище данных:**

LocalStorage для временного хранения данных на клиентской стороне.

IndexedDB для постоянного хранения данных на клиентской стороне.

**Взаимодействие между различными компонентами приложения**

**Загрузка данных:**

При загрузке страницы, данные имен загружаются из names.json и сохраняются в переменную namesData.

Функция openDatabase открывает базу данных IndexedDB.

Функция saveModelToIndexedDB сохраняет CSV данные в IndexedDB.

Функция loadModelFromIndexedDB загружает CSV данные из IndexedDB.

**Добавление и удаление столбцов:**

Функция addColumn добавляет новый столбец в форму.

Функция removeColumn удаляет столбец из формы.

Функция renumberColumns перенумеровывает столбцы после удаления.

**Генерация CSV данных:**

Функция generateCSV генерирует CSV данные на основе введенных параметров.

Функция generateData генерирует данные для каждого столбца в зависимости от типа данных и диапазона.

Функция escapeCSV экранирует специальные символы в CSV данных.

Функция displayCSVData отображает сгенерированные CSV данные в таблице.

**Сохранение и загрузка CSV данных:**

Функция saveCSV сохраняет CSV данные в файл.

Функция handleCSVFile обрабатывает загрузку CSV файла и сохраняет данные в localStorage или IndexedDB.

Функция clearTable очищает таблицу и сбрасывает форму.

**Анализ данных:**

Функция analyzeColumnData анализирует данные в столбце.

Функция analyzeWordFrequency анализирует частотность слов в текстовых данных.

Функция showAnalytics отображает аналитику данных, включая метрики и графики.

Функция buildHeatmap строит тепловую карту корреляции.

**Машинное обучение:**

Функция createAndTrainModel создает и обучает модель машинного обучения.

Функция showAnalytics использует обученную модель для прогнозирования данных.

**Экспорт данных:**

Функция exportToExcel экспортирует данные в Excel.

Функция exportToJson экспортирует данные в JSON.

**Поиск данных:**

Функция searchRows ищет совпадения в таблице на основе ввода пользователя.

**Пример взаимодействия:**

Пользователь добавляет столбцы через форму, выбирая тип данных и диапазон.

Пользователь нажимает кнопку "Генерация CSV", и данные генерируются на основе введенных параметров.

Сгенерированные данные отображаются в таблице и сохраняются в localStorage или IndexedDB.

Пользователь может загрузить CSV файл, и данные будут сохранены в localStorage или IndexedDB.

Пользователь может проанализировать данные, и результаты анализа отобразятся в виде метрик и графиков.

Пользователь может сохранить данные в файл, экспортировать их в Excel или JSON, а также открыть Jupyter Notebook или Google Colab с предустановленным кодом для анализа данных.

Таким образом, все компоненты приложения взаимодействуют друг с другом для обеспечения полного цикла работы с CSV данными: от их генерации и сохранения до анализа и экспорта.

**3.3. Интерфейс пользователя и основные элементы**

**Описание интерфейса пользователя**

Интерфейс пользователя CSV Generator разработан с учетом удобства и интуитивности использования. Он включает в себя следующие основные элементы:

**Форма для ввода параметров:**

Поля для ввода количества строк и имени файла.

Выбор типа индексов (без индексации строк, автоматические уникальные, пользовательские с пропусками).

**Форма для добавления столбцов:**

Поля для ввода названия столбца, типа данных и диапазона.

Кнопки для добавления и удаления столбцов.

Кнопки для выполнения действий:

Кнопки для создания CSV, сохранения CSV, загрузки CSV, экспорта в Excel и JSON, показа аналитики и очистки таблицы.

**Модальные окна:**

Модальное окно для отображения аналитики данных.

Модальные окна для состояния загрузки, сохранения данных.

**Таблица для отображения данных:**

Таблица для отображения сгенерированных или загруженных данных.

Поддержка пагинации для удобного просмотра больших объемов данных.

**Кнопки для выполнения действий:**

Позволяют пользователю выполнять различные действия, такие как создание CSV, сохранение данных, загрузка CSV файлов и т.д.

**Таблица для отображения данных:**

Отображает сгенерированные или загруженные данные в удобном табличном формате.

**3.4. Генерация данных**

Функция генерации данных позволяет пользователям создавать тестовые наборы данных с заданными параметрами и диапазонами. Пользователь может задавать количество строк, тип индексов, а также добавлять и удалять столбцы, задавая для каждого столбца тип данных и диапазон.

**Примеры использования различных типов данных и диапазонов**

Даты:

Диапазон: 2020.01.01-2023.01.01

Пример: Генерация дат в диапазоне от 1 января 2020 года до 1 января 2023 года.

**Время:**

Диапазон: 08:00:00-18:00:00

Пример: Генерация времени в диапазоне от 8:00 до 18:00.

**Числа (целые и дробные):**

Диапазон: 1-100 (целые числа) или 1.0-100.0 (дробные числа)

Пример: Генерация целых чисел в диапазоне от 1 до 100 или дробных чисел в диапазоне от 1.0 до 100.0.

**Строки:**

Диапазон: Список значений (например, А, Б, В)

Пример: Генерация строк из списка значений.

**Логические значения:**

Диапазон: true, false

Пример: Генерация логических значений true и false.

# 3.4.2 Валидация данных

**1. Функция validateInput:**

Эта функция проверяет корректность введенных данных для столбцов перед генерацией CSV.

**Проверка на пустое название столбца:**

if (!columnName) {

alert("Название столбца не может быть пустым.");

return false;

}

**Проверка на пустой тип данных:**

if (!dataType) {

alert("Тип данных не может быть пустым.");

return false;

}

**Проверка на пустой диапазон данных для определенных типов:**

if (dataType === "float\_range" || dataType === "integer\_range" || dataType === "range\_negative" || dataType === "choice" || dataType === "time" || dataType === "date") {

if (!dataRange) {

alert("Диапазон данных не может быть пустым.");

return false;

}

}

**Проверка корректности диапазона для числовых типов данных:**

if (dataType === "float\_range" || dataType === "integer\_range") {

const [start, end] = dataRange.split("-").map(Number);

if (isNaN(start) || isNaN(end) || start >= end) {

alert("Некорректный диапазон данных.");

return false;

}

}

**Проверка корректности диапазона для типа данных range\_negative:**

if (dataType === "range\_negative") {

const [startStr, endStr] = dataRange.split(",").map(str => str.trim());

const start = Number(startStr);

const end = Number(endStr);

if (isNaN(start) || isNaN(end) || start >= end) {

alert("Некорректный диапазон данных.");

return false;

}

}

**Проверка корректности диапазона для типа данных choice:**

if (dataType === "choice") {

const dataList = dataRange.split(",").map(str => str.trim());

if (dataList.length < 2) {

alert("Диапазон данных должен содержать как минимум два значения.");

return false;

}

}

**Проверка корректности формата даты:**

else if (dataType === "date") {

const dateRegex = /^\d{2}\.\d{2}\.\d{4}-\d{2}\.\d{2}\.\d{4}$/;

if (!dateRegex.test(dataRange)) {

alert("Некорректный формат даты. Используйте формат ДД.ММ.ГГГГ-ДД.ММ.ГГГГ");

return false;

}

const [startDateStr, endDateStr] = dataRange.split("-");

const startDate = new Date(startDateStr.split(".").reverse().join("-"));

const endDate = new Date(endDateStr.split(".").reverse().join("-"));

if (isNaN(startDate.getTime()) || isNaN(endDate.getTime()) || startDate >= endDate) {

alert("Некорректный диапазон дат.");

return false;

}

}

**Проверка корректности формата времени:**

if (dataType === "time") {

const timeRegex = /^\d{2}:\d{2}:\d{2}-\d{2}:\d{2}:\d{2}$/;

if (!timeRegex.test(dataRange)) {

alert("Некорректный формат времени. Используйте формат ЧЧ:ММ:СС-ЧЧ:ММ:СС.");

return false;

}

const [startTime, endTime] = dataRange.split("-");

const startParts = startTime.split(":").map(Number);

const endParts = endTime.split(":").map(Number);

if (startParts.length !== 3 || endParts.length !== 3) {

alert("Некорректный формат времени. Используйте формат ЧЧ:ММ:СС-ЧЧ:ММ:СС.");

return false;

}

const startTotalSeconds = startParts[0] \* 3600 + startParts[1] \* 60 + startParts[2];

const endTotalSeconds = endParts[0] \* 3600 + endParts[1] \* 60 + endParts[2];

if (startTotalSeconds >= endTotalSeconds) {

alert("Некорректный диапазон времени.");

return false;

}

}

**2. Валидация перед генерацией CSV:**

**Функция checkAndPrompt:**

Эта функция проверяет корректность введенных данных перед генерацией CSV.

**Проверка количества строк:**

const numRows = parseInt(document.getElementById("num\_rows").value, 10);

if (isNaN(numRows) || numRows <= 0) {

alert("Количество строк должно быть положительным числом.");

return false;

}

**Проверка имени файла:**

const filename = document.getElementById("filename").value;

if (!filename) {

alert("Имя файла не может быть пустым.");

return false;

}

**Проверка наличия столбцов:**

if (indexType === "none" && columnCount === 0) {

alert("Введите больше данных. Нажмите кнопку Добавить столбец");

return false;

}

**Проверка уникальности названий столбцов:**

const columnNames = [];

for (let i = 1; i <= columnCount; i++) {

const columnName = document.getElementById(`column\_name\_${i}`).value;

if (columnNames.includes(columnName)) {

const changeData = confirm(`"${columnName}" уже существует`);

if (changeData) {

return false;

}

}

columnNames.push(columnName);

}

**3. Валидация при загрузке CSV файла:**

**Функция handleCSVFile:**

Эта функция обрабатывает загрузку CSV файла и проверяет корректность данных.

**Проверка наличия ошибок в файле:**

csvData.forEach((row, rowIndex) => {

let rowErrors = [];

Object.keys(row).forEach((key) => {

if (row[key] === "" || row[key] === null || row[key] === undefined) {

rowErrors.push(`Пустое значение в столбце "${key}"`);

}

});

if (rowErrors.length > 0) {

hasErrors = true;

errorReport += `Строка ${rowIndex + 1}: ${rowErrors.join(", ")}\n`;

}

});

if (hasErrors) {

alert(errorReport);

}

**4. Валидация при анализе данных:**

**Функция analyzeColumnData:**

Эта функция анализирует данные в столбце и проверяет наличие пропущенных значений.

**Проверка наличия пропущенных значений:**

let missingValues = 0;

let nullValues = 0;

let undefinedValues = 0;

let nanValues = 0;

columnData.forEach(value => {

if (value === null) {

nullValues++;

} else if (value === undefined) {

undefinedValues++;

} else if (value === '' || value === 'NaN') {

nanValues++;

}

if (value === '' || value === null || value === undefined || value === 'NaN') {

missingValues++;

}

});

**5. Валидация при поиске данных:**

**Функция searchRows:**

Эта функция ищет совпадения в таблице на основе ввода пользователя и проверяет корректность введенных значений.

**Проверка наличия совпадений:**

const matchingRows = data.filter((row) => {

return searchValues.some((value) =>

row.some((cell) => cell.toLowerCase().includes(value))

);

});

if (matchingRows.length > 0) {

// Отображение найденных строк

} else {

const noResultsDiv = document.createElement("div");

noResultsDiv.textContent = "Совпадений не найдено.";

searchResultsDiv.appendChild(noResultsDiv);

}

**Заключение:**

Все эти меры валидации помогают предотвратить неправильный ввод данных и обеспечивают корректную работу приложения. Валидация происходит на различных этапах: при вводе данных для столбцов, перед генерацией CSV, при загрузке CSV файла, при анализе данных и при поиске данных. Это позволяет пользователю получать понятные сообщения об ошибках и исправлять их, что улучшает пользовательский опыт и надежность приложения.

**3.5. Загрузка и сохранение данных**

**Описание функции загрузки CSV файлов**

Функция загрузки CSV файлов позволяет пользователям загружать существующие CSV файлы для анализа. После загрузки файла данные отображаются в таблице, и пользователь может выполнять различные операции с ними, такие как анализ данных, визуализация и сохранение в других форматах.

Происходит сохранение данных в IndexedDB и localStorage в зависимости от объема.

Преимущества IndexedDB

Объем хранения:

IndexedDB: Может хранить большие объемы данных, ограниченные только доступным дисковым пространством.

localStorage: Ограничен обычно 5-10 МБ, что может быть недостаточно для больших CSV файлов.

Типы данных:

IndexedDB: Поддерживает хранение сложных объектов и структур данных, таких как массивы и объекты.

localStorage: Хранит данные только в виде строк, что требует сериализации и десериализации объектов.

Асинхронность:

IndexedDB: Все операции асинхронные, что позволяет избежать блокировки основного потока выполнения.

localStorage: Синхронные операции могут блокировать основной поток, особенно при работе с большими объемами данных.

Индексирование и запросы:

IndexedDB: Поддерживает индексирование и сложные запросы, что позволяет эффективно искать и извлекать данные.

localStorage: Не поддерживает индексирование и сложные запросы, что ограничивает возможности поиска и фильтрации данных.

Транзакции:

IndexedDB: Поддерживает транзакции, что обеспечивает целостность данных при выполнении нескольких операций.

localStorage: Не поддерживает транзакции, что может привести к проблемам целостности данных при одновременном выполнении нескольких операций.

Описание функции сохранения данных в различных форматах (CSV, Excel, JSON)

Функция сохранения данных позволяет пользователям сохранять сгенерированные или загруженные данные в различных форматах:

CSV:

Данные сохраняются в формате CSV, что позволяет легко обмениваться данными между различными системами и приложениями.

Excel:

Данные сохраняются в формате Excel, что позволяет использовать все возможности Excel для анализа и визуализации данных.

JSON:

Данные сохраняются в формате JSON, что позволяет легко обмениваться данными между веб-приложениями и серверами.

**3.6. Анализ данных**

Функция анализа данных позволяет пользователям выявлять пропуски, уникальные и повторяющиеся значения в данных, а также визуализировать данные с помощью графиков и тепловых карт. Это помогает лучше понять структуру данных и выявить возможные аномалии.

**Примеры анализа данных на наличие пропусков, уникальных и повторяющихся значений**

**Анализ пропусков:**

Анализ данных о продажах может выявить пропуски в данных о количестве проданных товаров. Это помогает очистить данные и обеспечить их корректный анализ. Например, если в данных о продажах отсутствуют значения для определенных дней, это может указывать на проблемы с учетом или загрузкой данных.

**Анализ уникальных и повторяющихся значений:**

Анализ данных о клиентах может выявить уникальные и повторяющиеся значения в данных о контактной информации. Это помогает определить, какие клиенты являются уникальными, а какие — повторяющимися. Например, если в базе данных клиентов обнаружены дублирующиеся записи, это может указывать на необходимость очистки данных для предотвращения дублирования.

**Визуализация данных с помощью графиков и тепловых карт**

**Графики:**

Визуализация данных о продажах в виде графиков может помочь выявить тренды и закономерности. Например, график продаж по дням может показать, в какие дни продажи были выше или ниже среднего, что поможет принять обоснованные решения по улучшению продаж.

**Тепловые карты:**

Тепловая карта корреляции — это графическое представление корреляционной матрицы, которое помогает визуализировать степень линейной зависимости между различными переменными в наборе данных. Вот что можно узнать из тепловой карты корреляции:

Сила корреляции:

Позитивная корреляция: Значения близкие к 1 указывают на сильную положительную корреляцию, что означает, что когда одна переменная увеличивается, другая также увеличивается.

Негативная корреляция: Значения близкие к -1 указывают на сильную отрицательную корреляцию, что означает, что когда одна переменная увеличивается, другая уменьшается.

Отсутствие корреляции: Значения близкие к 0 указывают на отсутствие линейной зависимости между переменными.

Мультиколлинеарность:

Высокие значения корреляции между независимыми переменными могут указывать на мультиколлинеарность, что может быть проблемой для некоторых моделей машинного обучения, таких как линейная регрессия.

Выбор признаков:

Тепловая карта корреляции может помочь в выборе наиболее значимых признаков для модели. Переменные с высокой корреляцией с целевой переменной могут быть полезны для предсказания.

Интерпретация данных:

Визуализация корреляций может помочь в понимании структуры данных и взаимосвязей между различными переменными. Это может быть полезно для генерации гипотез и дальнейшего анализа данных.

Пример интерпретации тепловой карты корреляции

Предположим, у вас есть набор данных с переменными A, B, C и D. Тепловая карта корреляции может показать следующие результаты:

A и B: Корреляция 0.9 (сильная положительная корреляция)

A и C: Корреляция -0.8 (сильная отрицательная корреляция)

B и C: Корреляция 0.1 (слабая корреляция)

C и D: Корреляция 0.7 (умеренная положительная корреляция)

Из этого можно сделать следующие выводы:

Переменные A и B сильно коррелируют друг с другом, что может указывать на мультиколлинеарность.

Переменные A и C имеют сильную отрицательную корреляцию, что означает, что они движутся в противоположных направлениях.

Переменные B и C практически не коррелируют.

Переменные C и D имеют умеренную положительную корреляцию, что может указывать на некоторую зависимость между ними.

Визуализация данных о корреляции между различными столбцами в виде тепловой карты может помочь выявить сильные и слабые корреляции между данными. Например, тепловая карта корреляции между столбцами "количество проданных товаров" и "рекламные расходы" может показать, насколько сильно рекламные расходы влияют на продажи.

**Поиск**

Функция поиска позволяет пользователям искать строки по данным и выводить результаты в процентном соотношении к остальным данным. Например, если пользователь ищет строки, содержащие определенное значение, функция поиска выведет количество найденных строк и их процентное соотношение к общему количеству строк. Это помогает оценить значимость найденных данных в контексте всего набора данных.

**Пример использования функции анализа данных**

**Анализ пропусков:**

Пользователь загружает CSV файл с данными о продажах.

Функция анализа данных выявляет пропуски в столбце "количество проданных товаров".

Пользователь получает отчет о количестве пропущенных значений и их процентном соотношении к общему количеству строк.

Анализ уникальных и повторяющихся значений:

Пользователь загружает CSV файл с данными о клиентах.

Функция анализа данных выявляет уникальные и повторяющиеся значения в столбце "контактная информация".

Пользователь получает отчет о количестве уникальных и повторяющихся значений, что помогает определить, какие клиенты являются уникальными, а какие — повторяющимися.

**Визуализация данных:**

Пользователь загружает CSV файл с данными о продажах.

Функция анализа данных строит график продаж по дням и тепловую карту корреляции между столбцами "количество проданных товаров" и "рекламные расходы".

Пользователь видит тренды продаж и корреляцию между рекламными расходами и продажами, что помогает принять обоснованные решения по улучшению продаж.

**Поиск данных:**

Пользователь вводит значение для поиска в данных о продажах.

Функция поиска выводит строки, содержащие это значение, и их процентное соотношение к общему количеству строк.

Пользователь получает отчет о количестве найденных строк и их значимости в контексте всего набора данных.

Таким образом, функция анализа данных предоставляет пользователям мощные инструменты для выявления пропусков, уникальных и повторяющихся значений, а также визуализации данных с помощью графиков и тепловых карт. Это помогает лучше понять структуру данных и выявить возможные аномалии, что способствует принятию обоснованных решений.

**3.7. Интеграция с машинным обучением**

Функция создания и обучения моделей машинного обучения позволяет пользователям создавать модели на основе загруженных данных и обучать их с использованием библиотеки TensorFlow.js. Пользователь увидит результат обучения, включая метрики точности и графики потерь, что помогает оценить качество модели и её способность делать точные прогнозы или классификации.

**Примеры использования моделей машинного обучения для анализа данных**

**Прогнозирование:**

Пример: Обучение модели для прогнозирования продаж на основе исторических данных о продажах. Пользователь может загрузить данные о продажах за предыдущие периоды и обучить модель для предсказания будущих продаж. Это помогает бизнесу планировать запасы и маркетинговые кампании более эффективно.

**Классификация:**

Пример: Обучение модели для классификации клиентов на основе их покупательского поведения. Пользователь может загрузить данные о покупках клиентов и обучить модель для определения сегментов клиентов, таких как "лояльные клиенты", "потенциальные клиенты" и "ушедшие клиенты". Это помогает бизнесу разрабатывать целевые маркетинговые стратегии для каждого сегмента.

**Регрессия:**

Пример: Обучение модели для регрессионного анализа зависимости между различными параметрами данных. Пользователь может загрузить данные о различных параметрах, таких как рекламные расходы и объем продаж, и обучить модель для определения, как изменения в рекламных расходах влияют на объем продаж. Это помогает бизнесу оптимизировать рекламные бюджеты и стратегии.

Можно выделить несколько примеров использования моделей машинного обучения для анализа данных. Вот как это может быть реализовано:

**1. Прогнозирование продаж**

**Пример:**

**Задача: Обучение модели для прогнозирования будущих продаж на основе исторических данных.**

**Решение:**

Загрузите исторические данные о продажах в формате CSV.

Используйте функцию createAndTrainModel для создания и обучения модели машинного обучения.

Модель будет обучена на исторических данных и сможет прогнозировать будущие продажи.

async function predictSales() {

let csvContent = localStorage.getItem("csvData");

if (!csvContent) {

csvContent = await loadModelFromIndexedDB();

if (!csvContent) {

alert("Сначала создайте или загрузите CSV данные.");

return;

}

}

const csvLines = csvContent.split("\n");

const headers = csvLines[0].split(",");

const data = csvLines.slice(1).map(line => line.split(",").map(Number));

const numRows = data.length;

const numCols = headers.length;

// Преобразование данных в тензор

const tensorData = tf.tensor2d(data, [numRows, numCols]);

// Нормализация данных

const mean = tensorData.mean(0, true);

const std = tensorData.sub(mean).square().mean(0, true).sqrt();

const normalizedData = tensorData.sub(mean).div(std);

// Разделение данных на обучающую и тестовую выборки

const splitIndex = Math.floor(normalizedData.shape[0] \* 0.8);

const xTrain = normalizedData.slice([0, 0], [splitIndex, normalizedData.shape[1] - 1]);

const yTrain = normalizedData.slice([0, normalizedData.shape[1] - 1], [splitIndex, 1]);

const xTest = normalizedData.slice([splitIndex, 0], [-1, normalizedData.shape[1] - 1]);

// Определение формы входных данных

const inputShape = [xTrain.shape[1]];

// Создание и обучение модели

await createAndTrainModel(inputShape, xTrain, yTrain);

// Прогнозирование на тестовой выборке

const predictions = model.predict(xTest);

// Денормализация прогнозов

const predictionsDenormalized = predictions.mul(std.slice([normalizedData.shape[1] - 1])).add(mean.slice([normalizedData.shape[1] - 1]));

// Вывод прогнозов

console.log('Прогнозы продаж:', predictionsDenormalized.arraySync());

// Отображение прогнозов в интерфейсе

const predictionsArray = predictionsDenormalized.arraySync();

const resultsDiv = document.getElementById('results');

resultsDiv.innerHTML = '';

const table = document.createElement("table");

const thead = document.createElement("thead");

const tbody = document.createElement("tbody");

const headerRow = document.createElement('tr');

headers.forEach(header => {

const th = document.createElement('th');

th.textContent = header;

headerRow.appendChild(th);

});

thead.appendChild(headerRow);

predictionsArray.forEach(prediction => {

const row = document.createElement('tr');

prediction.forEach(value => {

const td = document.createElement('td');

td.textContent = value.toFixed(2);

row.appendChild(td);

});

tbody.appendChild(row);

});

table.appendChild(thead);

table.appendChild(tbody);

resultsDiv.appendChild(table);

}

**2. Классификация клиентов**

**Пример:**

**Задача: Обучение модели для классификации клиентов на основе их покупательского поведения.**

**Решение:**

Загрузите данные о клиентах, включая историю покупок и демографические данные.

Используйте функцию createAndTrainModel для создания и обучения модели классификации.

Модель будет обучена на данных о клиентах и сможет классифицировать их на сегменты, такие как "лояльные клиенты", "потенциальные клиенты" и "ушедшие клиенты".

async function classifyCustomers() {

let csvContent = localStorage.getItem("csvData");

if (!csvContent) {

csvContent = await loadModelFromIndexedDB();

if (!csvContent) {

alert("Сначала создайте или загрузите CSV данные.");

return;

}

}

const csvLines = csvContent.split("\n");

const headers = csvLines[0].split(",");

const data = csvLines.slice(1).map(line => line.split(",").map(Number));

const numRows = data.length;

const numCols = headers.length;

const tensorData = tf.tensor2d(data, [numRows, numCols]);

const mean = tensorData.mean(0, true);

const std = tensorData.sub(mean).square().mean(0, true).sqrt();

const normalizedData = tensorData.sub(mean).div(std);

const splitIndex = Math.floor(normalizedData.shape[0] \* 0.8);

const xTrain = normalizedData.slice([0, 0], [splitIndex, normalizedData.shape[1] - 1]);

const yTrain = normalizedData.slice([0, normalizedData.shape[1] - 1], [splitIndex, 1]);

const xTest = normalizedData.slice([splitIndex, 0], [-1, normalizedData.shape[1] - 1]);

const inputShape = [xTrain.shape[1]];

await createAndTrainModel(inputShape, xTrain, yTrain);

const predictions = model.predict(xTest);

const predictedClasses = predictions.argMax(1).arraySync();

console.log('Классификация клиентов:', predictedClasses);

}

**3. Регрессионный анализ**

**Пример:**

**Задача: Обучение модели для регрессионного анализа зависимости между рекламными расходами и объемом продаж.**

**Решение:**

Загрузите данные о рекламных расходах и объеме продаж.

Используйте функцию createAndTrainModel для создания и обучения модели регрессии.

Модель будет обучена на данных и сможет прогнозировать объем продаж на основе рекламных расходов.

async function regressionAnalysis() {

let csvContent = localStorage.getItem("csvData");

if (!csvContent) {

csvContent = await loadModelFromIndexedDB();

if (!csvContent) {

alert("Сначала создайте или загрузите CSV данные.");

return;

}

}

const csvLines = csvContent.split("\n");

const headers = csvLines[0].split(",");

const data = csvLines.slice(1).map(line => line.split(",").map(Number));

const numRows = data.length;

const numCols = headers.length;

const tensorData = tf.tensor2d(data, [numRows, numCols]);

const mean = tensorData.mean(0, true);

const std = tensorData.sub(mean).square().mean(0, true).sqrt();

const normalizedData = tensorData.sub(mean).div(std);

const splitIndex = Math.floor(normalizedData.shape[0] \* 0.8);

const xTrain = normalizedData.slice([0, 0], [splitIndex, normalizedData.shape[1] - 1]);

const yTrain = normalizedData.slice([0, normalizedData.shape[1] - 1], [splitIndex, 1]);

const xTest = normalizedData.slice([splitIndex, 0], [-1, normalizedData.shape[1] - 1]);

const inputShape = [xTrain.shape[1]];

await createAndTrainModel(inputShape, xTrain, yTrain);

const predictions = model.predict(xTest);

const predictionsDenormalized = predictions.mul(std.slice([normalizedData.shape[1] - 1])).add(mean.slice([normalizedData.shape[1] - 1]));

console.log('Регрессионный анализ:', predictionsDenormalized);

}

Эти примеры демонстрируют, как можно использовать модели машинного обучения для анализа данных в различных сценариях.

Интеграция с машинным обучением предоставляет пользователям мощные инструменты для анализа данных и принятия обоснованных решений. Функция создания и обучения моделей позволяет пользователям прогнозировать будущие тренды, классифицировать данные и выявлять зависимости между различными параметрами. Это помогает бизнесу оптимизировать процессы, улучшать маркетинговые стратегии и повышать общую эффективность.

**Подробное описание функции создания и обучения моделей**

**Проверка корректности входных данных:**

Функция начинает с проверки, что yTrain.shape[1] не равно 0. Это важно, чтобы убедиться, что целевые данные имеют корректную форму.

**Определение архитектуры модели:**

Создается последовательная модель (tf.sequential()).

Добавляется первый полносвязный слой (dense) с 128 нейронами и активационной функцией ReLU. Этот слой принимает входные данные формы inputShape.

units: 128 — это количество нейронов в слое. Чем больше нейронов, тем больше информации модель может захватить, но это также может привести к переобучению, если данных недостаточно.

activation: 'relu' — это функция активации, которая помогает модели учиться. ReLU задает выход 0 для отрицательных значений и сохраняет положительные значения без изменений, что помогает избежать проблемы исчезающего градиента.

inputShape: [inputShape] — это форма входных данных, которую ожидает слой. Например, если ваши входные данные — это изображения размером 28x28 пикселей, то inputShape будет [28 \* 28].

Добавляется выходной полносвязный слой с количеством нейронов, равным количеству целевых переменных (yTrain.shape[1]), и линейной активационной функцией.

units: yTrain.shape[1] — количество нейронов в выходном слое соответствует количеству целевых переменных. Если вы решаете задачу регрессии и у вас есть, например, две целевые переменные, то yTrain.shape[1] будет равно 2.

activation: 'linear' — линейная функция активации используется на выходном слое, особенно в задачах регрессии, где требуется предсказать непрерывные значения.

**Компиляция модели:**

Модель компилируется с использованием оптимизатора Adam, функции потерь Mean Squared Error (MSE) и метрики MAE (Mean Absolute Error).

optimizer: 'adam' — оптимизатор Adam, который адаптивно изменяет скорость обучения для каждого параметра, что делает его эффективным для многих задач.

loss: 'meanSquaredError' — функция потерь, которая измеряет среднюю квадратичную ошибку между предсказанными и реальными значениями. Это распространенная функция потерь для задач регрессии.

metrics: ['meanAbsoluteError'] — метрика, которую мы будем отслеживать во время обучения. MAE (средняя абсолютная ошибка) показывает, насколько близки предсказанные значения к реальным.

**Обучение модели:**

Модель обучается на данных xTrain и yTrain в течение 10 эпох с размером пакета (batch size) 32.

В процессе обучения выводятся метрики точности и графики потерь для каждой эпохи, что позволяет пользователю отслеживать процесс обучения.

**Обработка ошибок:**

Если возникает ошибка в процессе создания или обучения модели, она выводится в консоль.

Функция createAndTrainModel позволяет создавать и обучать модели машинного обучения на основе загруженных данных. Она обеспечивает нормализацию данных, создание архитектуры модели, обучение модели и вывод метрик точности. Это делает её мощным инструментом для анализа данных и принятия обоснованных решений. Интеграция этой функции в ваше приложение позволит пользователям прогнозировать будущие тренды, классифицировать данные и выявлять зависимости между различными параметрами.

**3.8. Примеры использования приложения**

Приложение для анализа и обработки CSV данных имеет широкий спектр применения в различных сферах. Вот несколько примеров, как его можно использовать:

**1. Анализ продаж и маркетинга**

**Пример:**

**Задача: Определить эффективность маркетинговых кампаний.**

Решение: Загрузите данные о продажах и маркетинговых расходах. Используйте функцию анализа данных для выявления корреляции между рекламными расходами и объемом продаж. Обучите модель машинного обучения для прогнозирования будущих продаж на основе текущих маркетинговых стратегий.

**2. Управление клиентской базой**

**Пример:**

**Задача: Сегментация клиентов для персонализированных предложений.**

Решение: Загрузите данные о клиентах, включая историю покупок, демографические данные и поведение на сайте. Используйте функцию классификации для разделения клиентов на сегменты, такие как "лояльные клиенты", "потенциальные клиенты" и "ушедшие клиенты". Это поможет разработать целевые маркетинговые кампании для каждого сегмента.

**3. Финансовый анализ**

**Пример:**

**Задача: Прогнозирование финансовых показателей компании.**

Решение: Загрузите исторические финансовые данные компании, такие как доходы, расходы и прибыль. Используйте функцию регрессионного анализа для прогнозирования будущих финансовых показателей. Это поможет в планировании бюджета и принятии стратегических решений.

**4. Мониторинг производственных процессов**

**Пример:**

**Задача: Оптимизация производственных процессов.**

Решение: Загрузите данные о производственных процессах, включая время выполнения задач, использование ресурсов и качество продукции. Используйте функцию анализа данных для выявления узких мест и оптимизации производственных процессов. Обучите модель машинного обучения для прогнозирования производительности и качества продукции.

**5. Анализ данных здравоохранения**

**Пример:**

**Задача: Прогнозирование заболеваемости на основе медицинских данных.**

Решение: Загрузите данные о пациентах, включая медицинские показатели, историю болезней и демографические данные. Используйте функцию классификации для прогнозирования риска заболеваний. Это поможет в разработке профилактических мер и персонализированных планов лечения.

**6. Образовательный анализ**

**Пример:**

**Задача: Оценка успеваемости студентов.**

Решение: Загрузите данные о студентах, включая оценки, посещаемость и участие в учебных мероприятиях. Используйте функцию регрессионного анализа для выявления факторов, влияющих на успеваемость. Обучите модель машинного обучения для прогнозирования будущих оценок студентов и выявления тех, кто нуждается в дополнительной поддержке.

**7. Анализ транспортных данных**

**Пример:**

**Задача: Оптимизация маршрутов транспорта.**

Решение: Загрузите данные о маршрутах транспорта, включая время в пути, загруженность маршрутов и данные о пробках. Используйте функцию анализа данных для выявления оптимальных маршрутов и времени отправления. Обучите модель машинного обучения для прогнозирования загруженности маршрутов и оптимизации расписания.

**8. Анализ данных социальных сетей**

**Пример:**

**Задача: Оценка эффективности контента в социальных сетях.**

Решение: Загрузите данные о публикациях в социальных сетях, включая количество лайков, комментариев и репостов. Используйте функцию анализа данных для выявления наиболее эффективных типов контента и времени публикации. Обучите модель машинного обучения для прогнозирования вовлеченности аудитории и оптимизации контент-стратегии.

**9. Экологический мониторинг**

**Пример:**

**Задача: Анализ данных о качестве воздуха.**

Решение: Загрузите данные о качестве воздуха, включая уровни загрязняющих веществ и метеорологические данные. Используйте функцию анализа данных для выявления источников загрязнения и прогнозирования уровня загрязнения. Обучите модель машинного обучения для разработки мер по улучшению качества воздуха.

**10. Анализ данных о потреблении энергии**

**Пример:**

**Задача: Оптимизация потребления энергии.**

Решение: Загрузите данные о потреблении энергии, включая данные о потреблении электроэнергии, газа и воды. Используйте функцию анализа данных для выявления пиков потребления и факторов, влияющих на потребление энергии. Обучите модель машинного обучения для прогнозирования будущего потребления энергии и разработки мер по его оптимизации.

Эти примеры демонстрируют, как приложение для анализа и обработки CSV данных может быть использовано в различных сферах для решения конкретных задач. Интеграция с машинным обучением, анализ данных и визуализация предоставляют пользователям мощные инструменты для принятия обоснованных решений и оптимизации процессов.

**Пошаговые инструкции по использованию приложения**

**Генерация данных:**

Шаг 1: Введите количество строк и имя файла.

Шаг 2: Выберите тип индексов.

Шаг 3: Добавьте столбцы, задавая для каждого столбца название, тип данных и диапазон.

Шаг 4: Нажмите кнопку "Создать CSV".

**Загрузка данных:**

Шаг 1: Нажмите кнопку "Загрузить CSV" и выберите файл для загрузки.

Шаг 2: Данные отобразятся в таблице, и вы сможете выполнять различные операции с ними.

**Анализ данных:**

Шаг 1: Нажмите кнопку "Показать аналитику".

Шаг 2: В модальном окне вы увидите метрики анализа данных, такие как количество строк, столбцов, пропущенных значений и т.д.

Шаг 3: Визуализируйте данные с помощью графиков и тепловых карт.

**Сохранение данных:**

Шаг 1: Нажмите кнопку "Сохранить CSV", "Экспорт в Excel" или "Экспорт в Json" для сохранения данных в соответствующем формате.

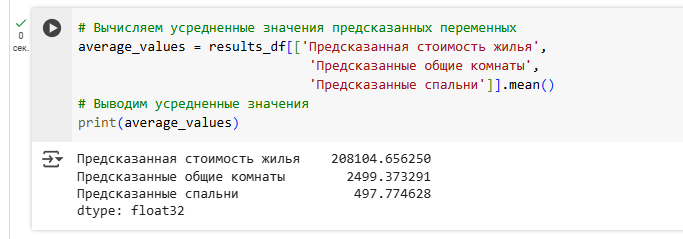
**Результаты и обсуждение**

**Анализ результатов**

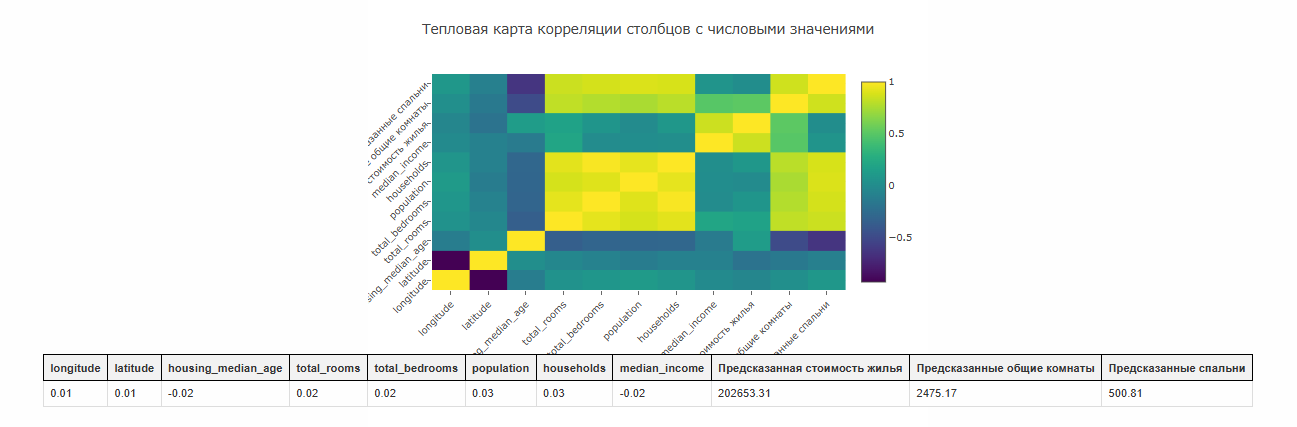
Результаты работы приложения показывают, что оно эффективно выполняет задачи генерации, анализа и управления CSV данными. Приложение позволяет пользователям легко создавать тестовые наборы данных, загружать и сохранять данные в различных форматах, анализировать данные и визуализировать результаты.

Анализ на одних и тех же данных

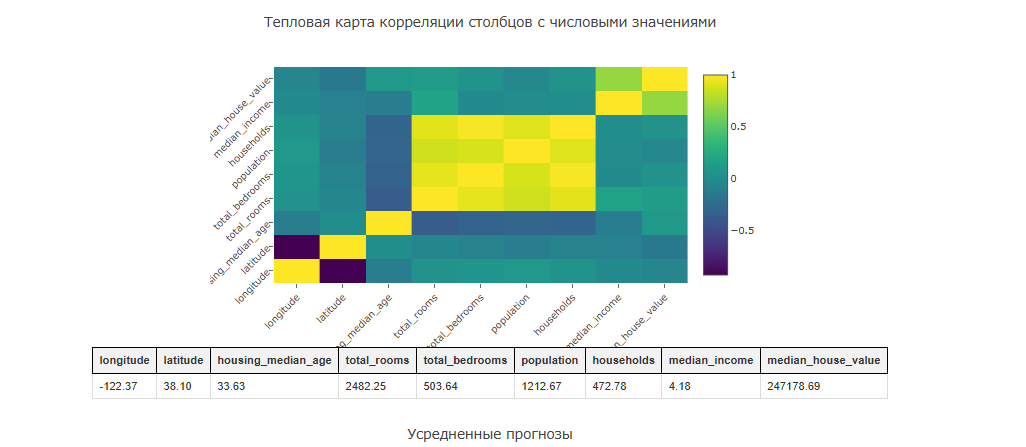
в Google Colab:



Анализ созданного файла на основе обучения



Результат аналитики исходных данных в приложении:



**4.1. Анализ результатов**

**Анализ результатов работы приложения**

Разработанное веб-приложение CSV Generator успешно выполняет задачи генерации, анализа и управления CSV данными. Оно предоставляет пользователям удобный и функциональный интерфейс для работы с данными, включая возможности генерации данных, загрузки и сохранения CSV файлов, анализа данных и визуализации результатов.

**Преимущества разработанного решения**

**Удобство использования:**

Интуитивно понятный интерфейс позволяет пользователям легко взаимодействовать с приложением и использовать его функции без необходимости глубоких технических знаний.

**Широкие возможности для генерации данных:**

Пользователи могут создавать тестовые наборы данных с заданными параметрами и диапазонами, что делает приложение полезным для тестирования алгоритмов и моделей машинного обучения.

**Поддержка различных форматов данных:**

Приложение поддерживает работу с данными в форматах CSV, Excel и JSON, что обеспечивает гибкость и удобство при обмене данными между различными системами и приложениями.

**Мощные инструменты для анализа данных:**

Приложение предоставляет возможности для анализа данных на наличие пропусков, уникальных и повторяющихся значений, а также визуализации данных с помощью графиков и тепловых карт.

**Интеграция с машинным обучением:**

Возможность создания и обучения моделей машинного обучения на основе загруженных данных с использованием библиотеки TensorFlow.js делает приложение мощным инструментом для анализа данных и прогнозирования.

**Недостатки разработанного решения**

При работе с очень большими объемами данных приложение может сталкиваться с проблемами производительности, такими как замедление интерфейса и увеличение времени обработки данных.

Необходимость наличия интернет-соединения для работы с машинным обучением:

Для использования функций машинного обучения требуется интернет-соединение, что может быть ограничением в условиях отсутствия доступа к интернету.

**4.2. Сравнение с существующими решениями**

Существует множество инструментов и приложений для анализа и обработки данных, каждый из которых имеет свои сильные и слабые стороны. Рассмотрим, как наше приложение сравнивается с некоторыми популярными решениями:

**1. Microsoft Excel / Google Sheets**

**Преимущества:**

Широкая функциональность: Поддержка различных формул и функций для анализа данных.

Интеграция: Легко интегрируется с другими продуктами Microsoft и Google.

Удобство использования: Интуитивно понятный интерфейс, знакомый многим пользователям.

**Недостатки:**

Ограниченные возможности машинного обучения: Требуется использование дополнительных инструментов или плагинов для машинного обучения.

Производительность: Может быть медленным при работе с большими объемами данных.

Ограниченные возможности визуализации: Базовые возможности визуализации данных.

**Сравнение с анализируемым приложением:**

Анализируемое приложение предоставляет встроенные возможности машинного обучения и более продвинутые методы визуализации данных.

Оптимизировано для работы с большими объемами данных.

**2. Pandas (Python)**

**Преимущества:**

Мощные возможности анализа данных: Широкий набор функций для анализа и преобразования данных.

Гибкость: Легко интегрируется с другими библиотеками Python, такими как NumPy, SciPy и Matplotlib.

Производительность: Оптимизировано для работы с большими объемами данных.

**Недостатки:**

Требует программирования: Необходимо знание Python для использования.

Отсутствие графического интерфейса: Требует использования дополнительных библиотек для визуализации данных.

**Сравнение с анализируемым приложением:**

Анализируемое приложение предоставляет графический интерфейс, что делает его доступным для пользователей без знаний программирования.

Встроенные возможности визуализации данных и машинного обучения.

**3. Tableau**

**Преимущества:**

Продвинутая визуализация данных: Широкий набор инструментов для создания интерактивных визуализаций.

Удобство использования: Интуитивно понятный интерфейс для создания дашбордов и отчетов.

Интеграция: Поддержка различных источников данных.

**Недостатки:**

Стоимость: Платное программное обеспечение, что может быть ограничением для некоторых пользователей.

Ограниченные возможности машинного обучения: Требуется использование дополнительных инструментов для машинного обучения.

**Сравнение с анализируемым приложением:**

Анализируемое приложение предоставляет встроенные возможности машинного обучения.

Бесплатное использование, открытый код, что делает его доступным для широкого круга пользователей.

**4. Power BI**

**Преимущества:**

Интеграция с Microsoft: Легко интегрируется с другими продуктами Microsoft.

Визуализация данных: Широкий набор инструментов для создания интерактивных визуализаций.

Удобство использования: Интуитивно понятный интерфейс для создания отчетов и дашбордов.

**Недостатки:**

Стоимость: Платное программное обеспечение, что может быть ограничением для некоторых пользователей.

Ограниченные возможности машинного обучения: Требуется использование дополнительных инструментов для машинного обучения.

**Сравнение с анализируемым приложением:**

Анализируемое приложение предоставляет встроенные возможности машинного обучения.

Бесплатное использование, что делает его доступным для широкого круга пользователей.

**5. Jupyter Notebook / Google Colab**

**Преимущества:**

Гибкость: Поддержка различных библиотек и инструментов для анализа данных и машинного обучения.

Облачные вычисления: Возможность использования облачных вычислительных ресурсов.

Документация: Легко документировать и делиться кодом и результатами анализа.

**Недостатки:**

Требует программирования: Необходимо знание Python для использования.

Отсутствие графического интерфейса: Требует использования кода для выполнения задач.

**Сравнение с анализируемым приложением:**

Анализируемое приложение предоставляет графический интерфейс, что делает его доступным для пользователей без знаний программирования.

Встроенные возможности визуализации данных и машинного обучения.

Моё приложение объединяет преимущества существующих решений, предоставляя мощные возможности анализа данных, машинного обучения и визуализации в удобном графическом интерфейсе. Оно оптимизировано для работы с большими объемами данных и предоставляет пользователям инструменты для принятия обоснованных решений. В отличие от многих существующих решений, наше приложение бесплатное и не требует знаний программирования, что делает его доступным для широкого круга пользователей.

**4.3. Возможные улучшения**

**Расширение функциональности**

**Поддержка других форматов данных:**

Добавление поддержки XML: Реализация функций для загрузки, анализа и экспорта данных в формате XML. Это позволит пользователям работать с более разнообразными источниками данных.

Поддержка JSON: Улучшение текущей функциональности для работы с JSON-данными, включая возможность загрузки и экспорта данных в формате JSON.

**Интеграция с облачными сервисами:**

Хранение данных в облаке: Интеграция с облачными сервисами, такими как Google Cloud Storage, AWS S3 или Azure Blob Storage, для хранения больших объемов данных. Это позволит пользователям работать с данными, не ограничиваясь локальными ресурсами.

Облачные вычисления: Использование облачных вычислительных ресурсов для обучения моделей машинного обучения, что позволит обрабатывать большие объемы данных быстрее и эффективнее.

**Реализация One-Hot Encoding:**

One-Hot Encoding для текстовых данных: Добавление функции для преобразования категориальных текстовых данных в числовой формат с помощью One-Hot Encoding. Это улучшит качество анализа текстовых данных и повысит точность моделей машинного обучения.

**Оптимизация производительности**

**Оптимизация алгоритмов:**

Улучшение алгоритмов генерации данных: Оптимизация алгоритмов для генерации данных, чтобы уменьшить время выполнения и потребление ресурсов.

Параллельная обработка данных: Использование многопоточности и параллельных вычислений для ускорения обработки больших объемов данных.

**Кэширование данных:**

Кэширование результатов анализа: Хранение результатов анализа данных в кэше для ускорения повторных запросов и уменьшения нагрузки на систему.

**Улучшение интерфейса пользователя**

**Добавление новых элементов интерфейса:**

Интерактивные графики: Добавление интерактивных графиков и диаграмм, которые позволяют пользователям взаимодействовать с данными в реальном времени.

Панели управления: Создание панелей управления для быстрого доступа к часто используемым функциям и настройкам.

**Улучшение существующих элементов:**

Удобство использования: Улучшение удобства использования интерфейса, включая интуитивно понятные меню и кнопки, а также улучшенную навигацию.

Адаптивный дизайн: Обеспечение адаптивного дизайна, который хорошо работает на различных устройствах, включая мобильные телефоны и планшеты.

**Подсказки и документация:**

Интерактивные подсказки: Добавление интерактивных подсказок и туториалов, которые помогут пользователям быстро освоить функциональность приложения.

Документация: Создание подробной документации по использованию приложения, включая примеры использования и часто задаваемые вопросы.

**Дополнительные функции**

**Интеграция с внешними API:**

API для анализа данных: Интеграция с внешними API для получения дополнительных данных, таких как погодные данные, финансовые показатели или данные социальных сетей.

API для машинного обучения: Использование внешних API для машинного обучения, таких как Google Cloud AI или AWS SageMaker, для улучшения качества моделей.

**Автоматизация процессов:**

Автоматическое обновление данных: Реализация функции автоматического обновления данных из внешних источников по расписанию.

Автоматическое обучение моделей: Автоматическое обучение моделей машинного обучения на новых данных без необходимости ручного вмешательства.

**Безопасность данных:**

Шифрование данных: Реализация шифрования данных для обеспечения безопасности при хранении и передаче данных.

Контроль доступа: Введение ролевого контроля доступа для ограничения доступа к данным и функциям приложения.

Эти улучшения и дополнения помогут сделать приложение более функциональным, производительным и удобным для пользователей. Расширение функциональности, оптимизация производительности и улучшение интерфейса пользователя позволят приложению лучше соответствовать потребностям пользователей и предоставлять более качественные результаты анализа данных.

**5.1. Выводы**

**Основные выводы по результатам работы**

Разработка веб-приложения CSV Generator была успешно завершена, и оно демонстрирует высокую функциональность и удобство использования. Приложение позволяет пользователям генерировать, анализировать и управлять CSV данными, предоставляя широкий спектр возможностей для работы с данными.

**Достижение поставленных целей и задач**

**Генерация данных:**

Цель: Разработать функцию для генерации данных с заданными параметрами и диапазонами.

Результат: Функция генерации данных успешно реализована. Пользователи могут создавать тестовые наборы данных с различными типами данных (даты, время, числа, строки и т.д.) и заданными диапазонами.

**Загрузка и сохранение данных:**

Цель: Обеспечить возможность загрузки CSV файлов для анализа и сохранения данных в различных форматах (CSV, Excel, JSON).

Результат: Функции загрузки и сохранения данных успешно реализованы. Пользователи могут загружать CSV файлы, анализировать их и сохранять данные в различных форматах.

**Анализ данных:**

Цель: Внедрить методы анализа данных, включая выявление пропусков, уникальных и повторяющихся значений, а также визуализацию данных.

Результат: Функции анализа данных успешно реализованы. Пользователи могут анализировать данные на наличие пропусков, уникальных и повторяющихся значений, а также визуализировать данные с помощью графиков и тепловых карт.

**Интеграция с машинным обучением:**

Цель: Создать и обучать модели машинного обучения на основе загруженных данных.

Результат: Функция создания и обучения моделей машинного обучения успешно реализована. Пользователи могут создавать модели на основе загруженных данных и обучать их с использованием библиотеки TensorFlow.js.

**Удобный интерфейс пользователя:**

Цель: Разработать интуитивно понятный интерфейс для взаимодействия с приложением.

Результат: Интерфейс пользователя успешно разработан. Пользователи могут легко взаимодействовать с приложением и использовать его функции без необходимости глубоких технических знаний.

**5.2. Рекомендации**

**Генерация данных:**

Используйте функцию генерации данных для создания тестовых наборов данных с заданными параметрами и диапазонами. Это полезно для тестирования алгоритмов и моделей машинного обучения.

Задавайте параметры генерации данных в соответствии с требованиями вашего проекта или задачи.

**Загрузка и сохранение данных:**

Используйте функцию загрузки CSV файлов для анализа существующих данных.

Сохраняйте сгенерированные или загруженные данные в форматах CSV, Excel или JSON в зависимости от ваших потребностей.

**Анализ данных:**

Используйте функции анализа данных для выявления пропусков, уникальных и повторяющихся значений. Это поможет очистить данные и обеспечить их корректный анализ.

Визуализируйте данные с помощью графиков и тепловых карт для лучшего понимания структуры данных и выявления закономерностей.

**Интеграция с машинным обучением:**

Используйте функцию создания и обучения моделей машинного обучения для анализа данных и прогнозирования.

Задавайте параметры модели, такие как количество эпох и размер батча, в соответствии с требованиями вашего проекта или задачи.

**6.1. Исходный код приложения**

**Полный исходный код приложения.**

**6.1. Комментарии и пояснения к коду**

**Комментарии и пояснения к коду**

**Общие комментарии**

**Структура HTML:**

HTML-документ содержит основные элементы, такие как заголовок, формы для ввода данных, кнопки для выполнения действий и таблицу для отображения данных.

Используются внешние библиотеки, такие как PapaParse для парсинга CSV, TensorFlow.js для машинного обучения, Chart.js, Plotly для визуализации данных и XLSX для работы с Excel файлами.

<!DOCTYPE html>

<html lang="ru">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Генератор CSV</title>

    <link rel="stylesheet" href="styles.css">

    <script src="script.js"></script>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/PapaParse/5.3.0/papaparse.min.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>

    <script src="https://cdn.jsdelivr.net/npm/papaparse"></script>

    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/xlsx/0.18.5/xlsx.full.min.js"></script>

    <link rel="preconnect" href="https://fonts.googleapis.com">

<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<link href="https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100..900;1,100..900&display=swap" rel="stylesheet">

</head>

<body>

    <h1>Генератор CSV</h1>

    <form id="csvForm">

        <div class="form-row">

            <!-- Поле ввода для количества строк -->

            <input type="number" id="num\_rows" name="num\_rows" placeholder="Количество строк" required

                title="Введите количество строк">

            <!-- Поле ввода для имени файла -->

            <input type="text" id="filename" name="filename" placeholder="Имя файла (без расширения)" required

                title="Введите имя файла">

            <!-- Выбор типа индексов -->

            <label for="indexType"></label>

            <select id="indexType" name="indexType" required>

                <option value="" disabled selected>Тип индексов</option>

                <option value="none" selected>Без индексации строк</option>

                <option value="auto">Автоматические уникальные</option>

                <option value="custom">Пользовательские с пропусками</option>

            </select>

            <!-- Контейнер для пользовательского диапазона индексов -->

            <div id="customIndexContainer">

                <label for="customIndexRange"></label>

                <input type="text" id="customIndexRange" style="display: none;" placeholder="Индексы (от-до)">

            </div>

        </div>

        <!-- Контейнер для столбцов -->

        <div id="columns"></div>

        <div class="buttons">

            <!-- Кнопки для управления таблицей -->

            <button type="button" class="button" onclick="addColumn()" title="Добавить столбец">Добавить столбец</button>

            <button type="button" class="button" onclick="generateCSV()" title="Создать CSV">Создать CSV</button>

            <button type="button" class="button" onclick="showAnalytics()" title="Показать аналитику">Показать аналитику</button>

            <input type="file" id="csvFileInput" accept=".csv" style="display: none">

            <button type="button" class="button" onclick="saveCSV()" title="Сохранить CSV">Сохранить CSV</button>

            <button type="button" class="button" onclick="document.getElementById('csvFileInput').click()" title="Загрузить CSV">Загрузить CSV</button>

            <button type="button" class="button" onclick="exportToExcel()" title="Экспортировать в Excel">Экспорт в Excel</button>

            <button type="button" class="button" onclick="exportToJson()" title="Экспортировать в Json">Экспорт в Json</button>

            <button type="button" class="button" onclick="clearTable()" title="Очистить таблицу">Очистить таблицу</button>

        </div>

    </form>

    <!-- Контейнер для отображения таблицы -->

    <div id="displaytable"></div>

    <!-- Контейнер для пагинации -->

    <div id="pagination\_\_container"></div>

    <!-- Контейнер для сообщений об ошибках -->

    <div id="errorMessages"></div>

    <!-- Сообщение о процессе -->

    <div id="processMessage">

        <div class="message-box"></div>

      </div>

    <!-- Контейнер для отображения загрузки -->

    <div id="loadingContainer">

        <div class="message-box"></div>

      </div>

    <!-- Модальное окно для аналитики -->

    <div id="analyticsModal" class="modal">

        <div class="modal-content">

            <span class="close" onclick="closeModal()">&times;</span>

            <h2>Аналитика таблицы</h2>

            <div id="analyticsContent"></div>

            <h2>Строки с NaN/пустыми значениями</h2>

            <div id="nanRowsContainer"></div> <!-- Контейнер для строк с NaN значениями -->

            <div id="graph" class="chartContainer"></div>

            <div id="chartContainer"></div>

            <div id="heatmap" class="chart-container"></div>

            <div id="results"></div>

            <div id="chart" class="chartContainer"></div>

            <!-- Форма ввода для пользователя -->

            <div id="userInputSection">

                <h3>Поиск строк по данным</h3>

                <input type="text" id="userInput"

                    placeholder="Введите данные через запятую (например, 1, 47, 3, 22, 56)">

                <button onclick="searchRows()">Поиск</button>

                <div id="searchResults"></div>

            </div>

            <div>

                <h3>Примеры для анализа данных:</h3>

                <button onclick="openJupyterNotebook()">Jupyter Notebook</button>

                <button onclick="openGoogleColab()">Google Colaboratory</button>

                <div class="hint" id="hintSection">

                    <h3>Как использовать файл в Jupyter Notebook или Google Colab:</h3>

                    <pre id="hintCode"></pre>

                </div>

            </div>

        </div>

        <div class="navigation">

            <h3>Навигация</h3>

            <ul>

                <li><a href="#analyticsContent">Общая аналитика</a></li>

                <li><a href="#nanRowsContainer">Строки с пустыми значениями</a></li>

                <li><a href="#graph">Графики данных</a></li>

                <li><a href="#heatmap">Тепловая карта корреляции</a></li>

                <li><a href="#results">Прогноз</a></li>

                <li><a href="#userInputSection">Поиск строки</a></li>

                <li><a href="#hintSection">Подсказки</a></li>

            </ul>

        </div>

    </div>

</body>

</html>

**Стили (CSS):**

Стили определяют внешний вид элементов интерфейса, таких как кнопки, таблицы и модальные окна.

Стили обеспечивают удобство использования и привлекательный внешний вид приложения.

@charset "UTF-8";

\* {

  margin: 0 auto;

  font-family: "Roboto", sans-serif;

}

.roboto {

  font-optical-sizing: auto;

  font-weight: normal;

  font-style: normal;

  font-variation-settings: "width" 100;

}

/\* Анимация плавного появления \*/

@keyframes fadeIn {

  from {

      opacity: 0;

  }

  to {

      opacity: 1;

  }

}

/\* Анимация изменения цвета \*/

@keyframes colorChange {

  0% {

      background-color: #f9f9f9;

  }

  50% {

      background-color: #d3d3d3;

  }

  100% {

      background-color: #f9f9f9;

  }

}

/\* Стили для контейнера загрузки \*/

#loadingContainer, #processMessage {

  display: none;

  position: fixed;

  top: 0;

  left: 0;

  width: 100%;

  height: 100%;

  background: rgba(0, 0, 0, 0.5) !important;

  justify-content: center !important;

  align-items: center !important;

  z-index: 1000 !important;

  animation: fadeIn 0.5s ease-in-out !important;

}

.message-box {

  padding: 20px;

  background: rgba(255, 255, 255, 0.8);

  color: black;

  border-radius: 10px;

  text-align: center;

}

/\* Стили для заголовка второго уровня \*/

h2 {

  text-align: center;

  margin: 30px 0px 30px 0px;

}

h1, h2 {

  animation: fadeIn 1s ease-in-out;

}

/\* Стили для таблицы \*/

table {

  width: 100%;

  border-collapse: collapse;

  animation: fadeIn 1s ease-in-out;

}

/\* Стили для заголовков и ячеек таблицы \*/

th, td {

  border: 1px solid black;

  padding: 8px;

  text-align: left;

}

/\* Стили для заголовков таблицы \*/

th {

  background-color: #f2f2f2;

}

/\* Стили для контейнера отображения таблицы \*/

#displaytable {

  margin-top: 20px;

}

/\* Стили для таблицы в контейнере отображения \*/

#displaytable table {

  width: 90%;

  border-collapse: collapse;

  margin-bottom: 20px;

}

/\* Стили для заголовков и ячеек таблицы в контейнере отображения \*/

#displaytable th, #displaytable td {

  border: 1px solid #ddd;

  padding: 8px;

  text-align: left;

}

/\* Стили для заголовков таблицы в контейнере отображения \*/

#displaytable th {

  background-color: #f2f2f2;

}

/\* Стили для четных строк таблицы в контейнере отображения \*/

#displaytable tr:nth-child(even) {

  background-color: #f9f9f9;

}

/\* Стиль для кнопок в контейнере кнопок на маленьких экранах \*/

.buttons {

  display: flex;

  justify-content: space-between;

  flex-direction: row;

  flex-wrap: wrap;

  gap: 11px;

  width: 100%;

  margin-top: 20px;

}

button {

  padding: 10px 20px;

  background-color: #1967d2;

  color: white;

  border: none;

  border-radius: 7px;

  cursor: pointer;

  transition: background-color 0.3s ease;

}

.buttons button {

  flex: 1;

  padding: 10px 20px;

  color: white;

  border: none;

  cursor: pointer;

  margin-right: 0px;

}

.buttons button:last-child {

  margin-right: 0;

}

.button:hover {

  background-color: #0793ff;

}

.button:active {

  background-color: #0793ff;

}

/\* Стили для таблицы аналитики \*/

.analytics-table {

  width: 100%;

  border-collapse: collapse;

}

/\* Стили для заголовков и ячеек таблицы аналитики \*/

.analytics-table th,

.analytics-table td {

  border: 1px solid #ddd;

  padding: 8px;

}

/\* Стили для заголовков таблицы аналитики \*/

.analytics-table th {

  background-color: #f2f2f2;

}

/\* Стили для подсказок \*/

.hint {

  margin-top: 20px;

  padding: 10px;

  border: 1px solid #ddd;

  background-color: #f9f9f9;

  animation: colorChange 2s infinite;

}

/\* Стили для модального окна \*/

.modal {

  display: none;

  position: fixed;

  z-index: 1;

  left: 0;

  top: 0;

  width: 100%;

  height: 100%;

  overflow: auto;

  background-color: rgba(0, 0, 0, 0.4);

}

.modal-content {

  position: relative;

  background-color: #fefefe;

  margin: 15% auto;

  padding: 20px;

  border: 1px solid #888;

  width: 90%;

  display: flex;

  flex-direction: column;

  align-items: center;

  animation: fadeIn 0.5s ease-in-out;

}

.close {

  position: absolute;

  color: #aaa;

  float: right;

  font-size: 28px;

  font-weight: bold;

  right: 10px;

}

.close:hover,

.close:focus {

  color: black;

  text-decoration: none;

  cursor: pointer;

}

/\* Стили для контейнера графиков \*/

.chart-container {

  justify-content: center;

}

#chartContainer {

  width: 100%;

  display: flex;

  flex-wrap: wrap;

  justify-content: center;

}

#chartContainer .text {

  font-family: Arial, sans-serif;

  font-size: 14px;

  color: #333;

}

#chartContainer canvas {

  width: 100%;

  margin: 10px;

}

/\* Стилизация осей \*/

#chartContainer .xtick,

#chartContainer .ytick {

  font-family: Arial, sans-serif;

  font-size: 12px;

  color: #333;

}

/\* Стилизация легенды \*/

#chartContainer .legend {

  font-family: Arial, sans-serif;

  font-size: 50px;

  color: #333;

}

/\* Стилизация подписей \*/

#chartContainer .annotation {

  font-family: Arial, sans-serif;

  font-size: 14px;

  color: #333;

}

#chart,

#graph {

  width: 100%;

  display: flex;

  flex-wrap: wrap;

  justify-content: center;

  margin-bottom: 30px;

}

/\* Графики занимают всю ширину \*/

#chart,

#graph,

#chartContainer canvas {

  width: 100%;

  flex-grow: 1;

}

/\* Текстовые элементы имеют единообразные стили \*/

.modal-content h2,

.modal-content p,

.modal-content th,

.modal-content td,

.modal-content .hint {

  font-family: Arial, sans-serif;

  font-size: 14px;

  color: #333;

  /\* background-color: #fff; \*/

}

.modal-content h2 {

  font-size: 18px;

}

.modal-content p {

  margin: 10px 0;

}

.modal-content th {

  background-color: #f2f2f2;

}

.modal-content td {

  border: 1px solid #ddd;

  padding: 8px;

}

.modal-content .hint {

  background-color: #f9f9f9;

  border: 1px solid #ddd;

  padding: 10px;

  margin-top: 20px;

}

/\* Стили для навигации \*/

.navigation {

  position: fixed;

  top: 25%;

  right: 10px;

  background: #447adb;

  border: 1px solid #ddd;

  padding: 10px;

  z-index: 1000;

}

.navigation h3 {

  margin-top: 0;

  margin-bottom: 15px;

  color: #ffffff;

}

.navigation ul {

  list-style-type: none;

  padding: 0;

}

.navigation ul li {

  margin-bottom: 10px;

}

.navigation ul li a {

  text-decoration: none;

  color: #ffffff;

}

.navigation ul li a:hover {

  text-decoration: underline;

}

.input,

.select {

  display: flex;

  flex-direction: column;

  padding: 8px;

  margin-bottom: 10px;

}

#csvForm {

  width: 90%;

  display: flex;

  flex-direction: column;

  align-items: center;

}

.form-row {

  display: flex;

  justify-content: space-between;

  width: 100%;

  margin-bottom: 10px;

  align-items: flex-end;

}

.form-row input,

.form-row select {

  flex: 1;

  margin-right: 10px;

  display: flex;

      flex-direction: column;

      padding: 8px;

      margin-bottom: 10px;

      border-radius: 7px 7px 7px 7px;

}

.label {

  display: flex;

  margin-bottom: 5px;

}

h1 {

  text-align: center;

  margin: 30px 0px 30px 0px;

}

/\* Стили для текстового поля с диапазоном \*/

.range {

  width: 100%;

  box-sizing: border-box;

  padding: 5px;

  overflow: hidden;

  resize: none;

}

/\* Стили для поля ввода и кнопки поиска \*/

#userInputSection {

  display: flex;

  flex-wrap: wrap;

  flex-direction: row;

  align-items: center;

  margin-bottom: 20px;

  align-items: flex-start;

  justify-content: center;

  width: 80%;

}

#userInputSection h3 {

  margin-bottom: 20px;

}

#userInput {

  width: 100%;

  padding: 10px;

  margin-bottom: 10px;

  font-size: 16px;

}

#userInputSection button {

  width: 100%;

  padding: 10px;

  font-size: 16px;

  margin-bottom: 30px;

}

#columns {

  width: 90%;

}

.column {

  width: 100%;

  margin-bottom: 20px;

}

.column-row {

  display: flex;

  flex-wrap: wrap;

  margin-bottom: 10px;

}

.row\_number {

  padding: 8px;

  background-color: #1967d2;

  color: white;

  border: none;

  display: flex;

  align-content: center;

  height: 19px;

  width: 19px;

  margin-right: 10px;

  justify-content: center;

}

.column-row input, .column-row select, .column-row textarea {

  margin-right: 10px;

  border-radius: 7px 7px 7px 7px;

}

.column-row input, .column-row select {

  flex: 1;

}

.row\_remove {

  background-color: red;

  color: white;

  border: none;

  padding: 5px;

  cursor: pointer;

  height: 34px;

  width: 34px;

}

.column textarea {

  width: 100%;

  margin-top: 10px;

}

select {

  color: #333;

}

option, title{

  color: #333;

}

/\* Стили для пагинации \*/

#pagination\_\_container {

  margin-top: 20px;

  display: flex;

  justify-content: center;

  width: 90%;

  align-items: center;

}

#pagination\_\_button {

  margin: 0 5px;

}

.pagination\_\_button {

  margin: 0 5px;

}

.percentage-info {

  font-weight: bold;

  color: #333;

  margin-bottom: 30px;

}

/\* Адаптивные стили \*/

/\* Медиа-запросы для адаптивности \*/

@media (max-width: 980px) {

  /\* Стиль для элементов ввода, выбора и кнопок в строке формы на маленьких экранах \*/

  .form-row input,

  .form-row select,

  .form-row button {

    flex: 1 1 100%;

    /\* Занимает всю доступную ширину \*/

  }

  .form-row {

    display: flex;

    flex-direction: row;

    flex-wrap: wrap;

  }

  .form-row input,

  .form-row select {

    flex: 1 1 100%;

    margin-right: 0px;

  }

}

@media (max-width: 480px) {

  .modal-content {

    width: 95%;

    padding: 10px;

  }

  .navigation {

    right: 2px;

    padding: 5px;

  }

  .navigation h3 {

    font-size: 14px;

  }

  .navigation ul li {

    margin-bottom: 6px;

  }

  .navigation ul li a {

    font-size: 12px;

  }

}

**Скрипты (JavaScript):**

JavaScript используется для реализации логики взаимодействия с пользователем, генерации данных, анализа данных и визуализации результатов.

Основные функции включают генерацию данных, загрузку и сохранение CSV файлов, анализ данных и интеграцию с машинным обучением.

let columnCount = 0;

let csvData = [];

let namesData = null;

let isCSVLoaded = false;

let columnMetadata = {};

let uploadedData = null;

let loadedColumns = [];

let loadedData = [];

let model;

const store\_name = 'csvDataStore';

// Загрузка данных имен

fetch("names.json")

.then((response) => {

if (!response.ok) {

throw new Error("Нет ответа сети " + response.statusText);

}

return response.json();

})

.then((data) => {

namesData = data;

})

.catch((error) => console.error("Ошибка загрузки данных имен:", error));

/\*\*

\* Открывает базу данных IndexedDB.

\* @returns {Promise<IDBDatabase>} - Возвращает промис, который разрешается в базу данных.

\*/

async function openDatabase() {

return new Promise((resolve, reject) => {

const request = indexedDB.open('csvDatabase', 1);

request.onupgradeneeded = (event) => {

const db = event.target.result;

if (!db.objectStoreNames.contains(store\_name)) {

db.createObjectStore(store\_name, { keyPath: 'id' });

}

};

request.onsuccess = () => resolve(request.result);

request.onerror = () => reject(request.error);

});

}

/\*\*

\* Сохраняет данные в IndexedDB.

\* @param {string} csvContentIndexedDB - CSV данные для сохранения.

\* @returns {Promise<void>} - Возвращает промис, который разрешается при успешном сохранении.

\*/

async function saveModelToIndexedDB(csvContentIndexedDB) {

const db = await openDatabase();

const transaction = db.transaction(store\_name, 'readwrite');

const store = transaction.objectStore(store\_name);

const request = store.put({ id: 'csvData', content: csvContentIndexedDB });

return new Promise((resolve, reject) => {

request.onsuccess = () => resolve();

request.onerror = () => reject(request.error);

});

}

/\*\*

\* Сохраняет данные в IndexedDB, разбивая их на части.

\* @param {string} csvContent - CSV данные для сохранения.

\*/

async function saveModelToIndexedDBInParts(csvContent) {

const chunkSize = 1 \* 1024 \* 1024; // 1 MB

const chunks = [];

for (let i = 0; i < csvContent.length; i += chunkSize) {

chunks.push(csvContent.slice(i, i + chunkSize));

}

const db = await openDatabase();

const transaction = db.transaction(store\_name, "readwrite");

const store = transaction.objectStore(store\_name);

for (let i = 0; i < chunks.length; i++) {

await new Promise((resolve, reject) => {

const request = store.put({ id: i, data: chunks[i] });

request.onsuccess = resolve;

request.onerror = reject;

});

}

await new Promise((resolve, reject) => {

transaction.oncomplete = resolve;

transaction.onerror = reject;

});

}

/\*\*

\* Загружает данные из IndexedDB.

\* @returns {Promise<string|null>} - Возвращает промис, который разрешается в загруженные данные или null, если данные не найдены.

\*/

async function loadModelFromIndexedDB() {

try {

const db = await openDatabase();

const transaction = db.transaction(store\_name, "readonly");

const store = transaction.objectStore(store\_name);

const request = store.getAll();

return new Promise((resolve, reject) => {

request.onsuccess = () => {

if (request.result && request.result.length > 0) {

// Объединяем части данных из IndexedDB

const csvContent = request.result

.sort((a, b) => a.id - b.id)

.map((item) => item.data)

.filter((data) => typeof data === "string") // Фильтруем только строки

.join("");

console.log("Загруженные данные из IndexedDB:", csvContent);

resolve(csvContent);

} else {

resolve(null);

}

};

request.onerror = () => reject(request.error);

});

} catch (error) {

console.error("Ошибка загрузки данных из IndexedDB:", error);

return null;

}

}

/\*\*

\* Очищает базу данных IndexedDB и удаляет её.

\* @returns {Promise<void>} - Возвращает промис, который разрешается после успешной очистки и удаления базы данных.

\*/

async function clearIndexedDB() {

const dbName = 'csvDatabase';

// Открыть базу данных

const db = await openDatabase();

// Очистить хранилище объектов

const transaction = db.transaction(store\_name, 'readwrite');

const store = transaction.objectStore(store\_name);

const clearRequest = store.clear();

await new Promise((resolve, reject) => {

clearRequest.onsuccess = () => resolve();

clearRequest.onerror = () => reject(clearRequest.error);

});

// Закрыть соединение с базой данных

db.close();

// Удалить базу данных

const deleteRequest = indexedDB.deleteDatabase(dbName);

deleteRequest.onsuccess = () => {

console.log('База данных успешно удалена');

};

deleteRequest.onerror = () => {

console.error('Ошибка при удалении базы данных:', deleteRequest.error);

};

deleteRequest.onblocked = () => {

console.warn('Операция удаления заблокирована. Повторная попытка через 1 секунду...');

setTimeout(() => {

indexedDB.deleteDatabase(dbName);

}, 1000);

};

}

/\*\*

\* Добавляет новый столбец в форму.

\* @param {string} columnName - Название столбца.

\* @param {string} dataType - Тип данных.

\* @param {string} dataRange - Диапазон данных.

\*/

function addColumn(columnName = "", dataType = "", dataRange = "") {

columnCount++;

const newColumn = document.createElement("div");

newColumn.className = "column";

newColumn.innerHTML = `

<div class="column-row">

<div class='row\_number'>${columnCount}</div>

<input type="text" id="column\_name\_${columnCount}" name="column\_names" placeholder="Название столбца ${columnCount}" value="${columnName}" required title="Введите название столбца">

<select id="data\_type\_${columnCount}" name="data\_type\_column\_name\_${columnCount}" required title="Выберите тип данных">

<option value="" disabled ${dataType ? "" : "selected"}>Тип данных</option>

<option value="date" ${dataType === "date" ? "selected" : ""}>date</option>

<option value="time" ${dataType === "time" ? "selected" : ""}>time</option>

<option value="float\_range" ${dataType === "float\_range" ? "selected" : ""}>float\_range</option>

<option value="integer\_range" ${dataType === "integer\_range" ? "selected" : ""}>integer\_range</option>

<option value="range\_negative" ${dataType === "range\_negative" ? "selected" : ""}>range\_negative</option>

<option value="choice" ${dataType === "choice" ? "selected" : ""}>choice</option>

<option value="boolean" ${dataType === "boolean" ? "selected" : ""}>boolean</option>

<option value="name" ${dataType === "name" ? "selected" : ""}>name</option>

<option value="phone" ${dataType === "phone" ? "selected" : ""}>phone</option>

<option value="email" ${dataType === "email" ? "selected" : ""}>email</option>

<option value="random" ${dataType === "random" ? "selected" : ""}>random</option>

${!isCSVLoaded ? `<option value="fromloadcsv" ${dataType === "fromloadcsv" ? "selected" : ""}>fromloadcsv</option>` : ''}

</select>

<button type="button" class="row\_remove" onclick="removeColumn(${columnCount})" title="Удалить столбец">X</button>

<textarea id="data\_range\_${columnCount}" class="range" name="data\_range\_column\_name\_${columnCount}" placeholder="Диапазон" title="Введите диапазон данных">${dataRange}</textarea>

</div>

`;

document.getElementById("columns").appendChild(newColumn);

updateDataRangeVisibility(columnCount);

styleSelect(columnCount);

adjustTextareaHeight(columnCount);

document.getElementById(`data\_type\_${columnCount}`).addEventListener("change", function () {

const dataType = this.value;

const dataRangeInput = document.getElementById(`data\_range\_${columnCount}`);

switch (dataType) {

case "date":

dataRangeInput.placeholder = "ДД.ММ.ГГГГ-ДД.ММ.ГГГГ";

break;

case "time":

dataRangeInput.placeholder = "ЧЧ:ММ:СС-ЧЧ:ММ:СС";

break;

case "float\_range":

case "integer\_range":

dataRangeInput.placeholder = "От-до (например, 1-10)";

break;

case "range\_negative":

dataRangeInput.placeholder = "От, до (например, -10, 5 или -1.56, 0)";

break;

case "choice":

dataRangeInput.placeholder = "Значение1, Значение2, Значение3 и т.д.";

break;

default:

dataRangeInput.placeholder = "";

}

});

const dataTypeSelect = document.getElementById(`data\_type\_${columnCount}`);

if (dataTypeSelect) {

updateDataRangeVisibility(columnCount);

}

}

/\*\*

\* Удаляет столбец из формы.

\* @param {number} columnIndex - Индекс столбца.

\*/

function removeColumn(columnIndex) {

const column = document.getElementById(`column\_name\_${columnIndex}`).parentElement.parentElement;

column.remove();

columnCount--;

renumberColumns();

}

/\*\*

\* Перенумеровывает столбцы после удаления.

\*/

function renumberColumns() {

const columns = document.querySelectorAll(".column-row");

columns.forEach((column, index) => {

const columnNameInput = column.querySelector('input[name="column\_names"]');

const dataTypeSelect = column.querySelector('select[name^="data\_type\_column\_name\_"]');

const dataRangeTextarea = column.querySelector('textarea[name^="data\_range\_column\_name\_"]');

const rowNumber = column.querySelector(".row\_number");

if (columnNameInput && dataTypeSelect && dataRangeTextarea && rowNumber) {

columnNameInput.id = `column\_name\_${index + 1}`;

columnNameInput.placeholder = `Название столбца ${index + 1}`;

dataTypeSelect.id = `data\_type\_${index + 1}`;

dataTypeSelect.name = `data\_type\_column\_name\_${index + 1}`;

dataRangeTextarea.id = `data\_range\_${index + 1}`;

dataRangeTextarea.name = `data\_range\_column\_name\_${index + 1}`;

rowNumber.textContent = index + 1;

const removeButton = column.querySelector(".row\_remove");

removeButton.onclick = function () {

removeColumn(index + 1);

};

} else {

console.error(`Один или несколько элементов не найдены для колонки ${index + 1}`);

}

});

}

/\*\*

\* Обновляет видимость поля ввода диапазона данных в зависимости от типа данных.

\* @param {number} columnNumber - Номер столбца.

\*/

function updateDataRangeVisibility(columnNumber) {

const dataTypeSelect = document.getElementById(`data\_type\_${columnNumber}`);

const dataRangeTextarea = document.getElementById(`data\_range\_${columnNumber}`);

if (dataTypeSelect && dataRangeTextarea) {

dataTypeSelect.addEventListener("change", function () {

if (dataTypeSelect.value === "float\_range" || dataTypeSelect.value === "date" || dataTypeSelect.value === "integer\_range" || dataTypeSelect.value === "range\_negative" || dataTypeSelect.value === "choice" || dataTypeSelect.value === "time") {

dataRangeTextarea.style.display = "block";

} else {

dataRangeTextarea.style.display = "none";

}

});

}

}

/\*\*

\* Обновляет стиль выбора типа данных.

\* @param {number} columnNumber - Номер столбца.

\*/

function styleSelect(columnNumber) {

const dataTypeSelect = document.getElementById(`data\_type\_${columnNumber}`);

dataTypeSelect.addEventListener("change", function () {

if (dataTypeSelect.value === "") {

dataTypeSelect.style.color = "darkgray";

} else {

dataTypeSelect.style.color = "black";

}

});

if (dataTypeSelect.value === "") {

dataTypeSelect.style.color = "darkgray";

} else {

dataTypeSelect.style.color = "black";

}

}

/\*\*

\* Регулирует высоту текстового поля в зависимости от введенного текста.

\* @param {number} columnNumber - Номер столбца.

\*/

function adjustTextareaHeight(columnNumber) {

const textarea = document.getElementById(`data\_range\_${columnNumber}`);

textarea.addEventListener("input", function () {

this.style.height = "auto";

this.style.height = this.scrollHeight + "px";

});

}

/\*\*

\* Экранирует специальные символы в строке для CSV.

\* @param {string} value - Значение для экранирования.

\* @returns {string} - Экранированное значение.

\*/

function escapeCSV(value) {

if (typeof value === 'string') {

value = value.replace(/"/g, '""');

if (value.includes(',') || value.includes('\n') || value.includes('"')) {

value = `"${value}"`;

}

}

return value;

}

/\*\*

\* Генерирует CSV данные на основе введенных параметров.

\*/

/\*\*

\* Генерирует CSV данные на основе введенных параметров.

\*/

async function generateCSV() {

if (uploadedData) {

// Если файл был загружен, используем его данные и добавляем новые столбцы

const csvData = generateCSVDataWithLoadedData();

displayCSVData(csvData);

} else {

if (!checkAndPrompt()) {

return;

}

const processMessage = document.getElementById("processMessage");

const messageBox = processMessage.querySelector(".message-box");

// Устанавливаем текст сообщения

messageBox.textContent = "Создаем данные... Пожалуйста, подождите.";

// Отображаем контейнер

processMessage.style.display = "flex";

const numRows = parseInt(document.getElementById("num\_rows").value, 10);

const filename = document.getElementById("filename").value + ".csv";

const indexType = document.getElementById("indexType").value;

let indexes;

if (indexType === "auto") {

indexes = Array.from({ length: numRows }, (\_, i) => i + 1);

} else if (indexType === "custom") {

const customIndexRange = document.getElementById("customIndexRange").value;

const [min, max] = customIndexRange.split("-").map(Number);

indexes = generateCustomIndexes(min, max, numRows);

} else {

indexes = Array(numRows).fill(null);

}

const columns = [];

const data = [];

document.getElementById("displaytable").style.display = "block";

for (let i = 1; i <= columnCount; i++) {

const columnName = document.getElementById(`column\_name\_${i}`).value;

const dataType = document.getElementById(`data\_type\_${i}`).value;

const dataRange = document.getElementById(`data\_range\_${i}`).value;

if (!validateInput(columnName, dataType, dataRange)) {

console.log(`Ошибка при генерации данных для столбца ${columnName}`);

processMessage.style.display = "none"; // Скрываем сообщение после завершения сохранения

return;

}

columns.push(columnName);

const generatedData = generateData(dataType, dataRange, numRows, columnName);

data.push(generatedData);

columnMetadata[columnName] = { dataType, dataRange };

// Обновление поля ввода диапазона

// const uniqueValues = [...new Set(generatedData)];

// document.getElementById(`data\_range\_${i}`).value = uniqueValues.join(", ");

// Обновление типа данных на "choice", если диапазон введен через запятую

// if (dataRange.includes(",")) {

// const values = dataRange.split(",").map(value => value.trim());

// const isRange = values.every(value => !isNaN(value));

// if (isRange) {

// document.getElementById(`data\_type\_${i}`).value = "choice";

// }

// }

}

if (indexType !== "none") {

data.unshift(indexes);

columns.unshift("index");

}

const csvContent = [columns.map(escapeCSV).join(",")]

.concat(data[0].map((\_, i) => data.map(col => escapeCSV(col[i] || "")).join(",")))

.join("\n");

// Сохранение данных в IndexedDB, если размер превышает ограничения localStorage

if (csvContent.length > 5 \* 1024 \* 1024) {

// 5 MB

await saveModelToIndexedDBInParts(csvContent);

} else {

localStorage.setItem("csvData", csvContent);

}

displayCSVData(csvContent);

console.log("Сгенерированные CSV данные:", csvContent);

console.log("Сгенерированные данные:", data);

updateHintSection(filename);

processMessage.style.display = "none"; // Скрываем сообщение после завершения сохранения

}

}

/\*\*

\* Генерирует CSV данные с загруженными данными.

\* @returns {string} - Сгенерированные CSV данные.

\*/

function generateCSVDataWithLoadedData() {

const columns = [...loadedColumns];

const data = [...loadedData];

for (let i = 1; i <= columnCount; i++) {

const columnName = document.getElementById(`column\_name\_${i}`).value;

const dataType = document.getElementById(`data\_type\_${i}`).value;

const dataRange = document.getElementById(`data\_range\_${i}`).value;

if (!validateInput(columnName, dataType, dataRange)) {

console.log(`Ошибка при генерации данных для столбца ${columnName}`);

return;

}

columns.push(columnName);

const generatedData = generateData(dataType, dataRange, loadedData.length, columnName);

for (let j = 0; j < loadedData.length; j++) {

data[j].push(generatedData[j]);

}

columnMetadata[columnName] = { dataType, dataRange };

// Обновление поля ввода диапазона

const uniqueValues = [...new Set(generatedData)];

document.getElementById(`data\_range\_${columnCount}`).value = uniqueValues.join(", ");

}

const csvContent = [columns.map(escapeCSV).join(",")]

.concat(data.map(row => row.map(escapeCSV).join(",")))

.join("\n");

localStorage.setItem("csvData", csvContent);

displayCSVData(csvContent);

console.log("Сгенерированные CSV данные:", csvContent);

console.log("Сгенерированные данные:", data);

// Обновляем секцию подсказок

updateHintSection(document.getElementById('fileName').value);

return csvContent;

}

/\*\*

\* Генерирует пользовательские индексы с пропусками.

\* @param {number} min - Минимальное значение индекса.

\* @param {number} max - Максимальное значение индекса.

\* @param {number} numRows - Количество строк.

\* @returns {Array} - Массив сгенерированных индексов.

\*/

function generateCustomIndexes(min, max, numRows) {

const indexes = [];

let currentIndex = min;

for (let i = 0; i < numRows; i++) {

if (Math.random() < 0.1) {

indexes.push(""); // Используем пустую строку для пропущенных значений

} else {

if (currentIndex <= max) {

indexes.push(currentIndex);

currentIndex++;

} else {

indexes.push(""); // Используем пустую строку для пропущенных значений

}

}

}

return indexes;

}

/\*\*

\* Генерирует данные для столбца на основе типа данных и диапазона.

\* @param {string} dataType - Тип данных.

\* @param {string} dataRange - Диапазон данных.

\* @param {number} numRows - Количество строк.

\* @param {string} columnName - Название столбца.

\* @returns {Array} - Массив сгенерированных данных.

\*/

function generateData(dataType, dataRange, numRows, columnName) {

if (dataType === "date") {

const [startDateStr, endDateStr] = dataRange.split("-");

const startDate = new Date(startDateStr.split(".").reverse().join("-"));

const endDate = new Date(endDateStr.split(".").reverse().join("-"));

if (isNaN(startDate.getTime()) || isNaN(endDate.getTime())) {

alert("Некорректный формат даты. Используйте формат ГГГГ.ММ.ДД-ГГГГ.ММ.ДД.");

return [];

}

const delta = (endDate - startDate) / (1000 \* 60 \* 60 \* 24);

return Array.from({ length: numRows }, () => {

const randomDate = new Date(startDate.getTime() + Math.random() \* delta \* 24 \* 60 \* 60 \* 1000);

return randomDate.toLocaleDateString("ru-RU");

});

} else if (dataType === "time") {

const [startTime, endTime] = dataRange.split("-");

return Array.from({ length: numRows }, () => generateRandomTime(startTime, endTime));

} else if (dataType === "float\_range") {

const [start, end] = dataRange.split("-").map(Number);

return Array.from({ length: numRows }, () => Math.random() \* (end - start) + start);

} else if (dataType === "integer\_range") {

const [start, end] = dataRange.split("-").map(Number);

return Array.from({ length: numRows }, () => Math.floor(Math.random() \* (end - start + 1)) + start);

} else if (dataType === "range\_negative") {

const [startStr, endStr] = dataRange.split(",").map((str) => str.trim());

const start = Number(startStr);

const end = Number(endStr);

return Array.from({ length: numRows }, () => Math.floor(Math.random() \* (end - start + 1)) + start);

} else if (dataType === "choice") {

const dataList = dataRange.split(",");

return Array.from({ length: numRows }, () => dataList[Math.floor(Math.random() \* dataList.length)]);

} else if (dataType === "boolean") {

return Array.from({ length: numRows }, () => Math.random() < 0.5 ? "true" : "false");

} else if (dataType === "name") {

if (!namesData) {

alert("Данные имен не загружены.");

return [];

}

const { maleFirstNames, femaleFirstNames, maleLastNames, femaleLastNames } = namesData;

return Array.from({ length: numRows }, () => {

const isMale = Math.random() < 0.5;

const firstName = isMale ? maleFirstNames[Math.floor(Math.random() \* maleFirstNames.length)] : femaleFirstNames[Math.floor(Math.random() \* femaleFirstNames.length)];

const lastName = isMale ? maleLastNames[Math.floor(Math.random() \* maleLastNames.length)] : femaleLastNames[Math.floor(Math.random() \* femaleLastNames.length)];

return `${firstName} ${lastName}`;

});

} else if (dataType === "phone") {

return Array.from({ length: numRows }, () => generatePhoneNumber());

} else if (dataType === "email") {

return Array.from({ length: numRows }, () => generateEmail());

} else if (dataType === "random") {

return Array.from({ length: numRows }, () => Math.random());

} else if (dataType === "fromloadcsv") {

return Array.from({ length: numRows }, (\_, i) => csvData[i][columnName]);

} else if (dataType === "") {

alert("Сначала создайте CSV данные.");

return [];

} else {

throw new Error("Неверный формат данных");

}

}

/\*\*

\* Генерирует случайное время в заданном диапазоне.

\* @param {string} startTime - Начальное время.

\* @param {string} endTime - Конечное время.

\* @returns {string} - Сгенерированное время.

\*/

function generateRandomTime(startTime, endTime) {

const [startHours, startMinutes, startSeconds] = startTime.split(":").map(Number);

const [endHours, endMinutes, endSeconds] = endTime.split(":").map(Number);

const startTotalSeconds = startHours \* 3600 + startMinutes \* 60 + startSeconds;

const endTotalSeconds = endHours \* 3600 + endMinutes \* 60 + endSeconds;

const randomTotalSeconds = Math.floor(Math.random() \* (endTotalSeconds - startTotalSeconds + 1)) + startTotalSeconds;

const hours = Math.floor(randomTotalSeconds / 3600);

const minutes = Math.floor((randomTotalSeconds % 3600) / 60);

const seconds = randomTotalSeconds % 60;

return `${String(hours).padStart(2, '0')}:${String(minutes).padStart(2, '0')}:${String(seconds).padStart(2, '0')}`;

}

/\*\*

\* Генерирует случайный номер телефона.

\* @returns {string} - Сгенерированный номер телефона.

\*/

function generatePhoneNumber() {

const phoneNumber = "+7" + Math.floor(Math.random() \* 10000000000).toString().padStart(10, "0");

return phoneNumber;

}

/\*\*

\* Генерирует случайный email.

\* @returns {string} - Сгенерированный email.

\*/

function generateEmail() {

const domains = ["gmail.com", "yahoo.com", "hotmail.com", "outlook.com"];

const username = Math.random().toString(36).substring(2, 10);

const domain = domains[Math.floor(Math.random() \* domains.length)];

return `${username}@${domain}`;

}

/\*\*

\* Валидирует ввод для столбца.

\* @param {string} columnName - Название столбца.

\* @param {string} dataType - Тип данных.

\* @param {string} dataRange - Диапазон данных.

\* @returns {boolean} - Результат валидации.

\*/

function validateInput(columnName, dataType, dataRange) {

if (!columnName) {

alert("Название столбца не может быть пустым.");

return false;

}

if (!dataType) {

alert("Тип данных не может быть пустым.");

return false;

}

if (dataType === "float\_range" || dataType === "integer\_range" || dataType === "range\_negative" || dataType === "choice" || dataType === "time" || dataType === "date") {

if (!dataRange) {

alert("Диапазон данных не может быть пустым.");

return false;

}

if (dataType === "float\_range" || dataType === "integer\_range") {

const [start, end] = dataRange.split("-").map(Number);

if (isNaN(start) || isNaN(end) || start >= end) {

alert("Некорректный диапазон данных.");

return false;

}

} else if (dataType === "range\_negative") {

const [startStr, endStr] = dataRange.split(",").map(str => str.trim());

const start = Number(startStr);

const end = Number(endStr);

if (isNaN(start) || isNaN(end) || start >= end) {

alert("Некорректный диапазон данных.");

return false;

}

} else if (dataType === "choice") {

const dataList = dataRange.split(",").map(str => str.trim());

if (dataList.length < 2) {

alert("Диапазон данных должен содержать как минимум два значения.");

return false;

}

} else if (dataType === "date") {

const dateRegex = /^\d{2}\.\d{2}\.\d{4}-\d{2}\.\d{2}\.\d{4}$/;

if (!dateRegex.test(dataRange)) {

alert("Некорректный формат даты. Используйте формат ДД.ММ.ГГГГ-ДД.ММ.ГГГГ");

return false;

}

const [startDateStr, endDateStr] = dataRange.split("-");

const startDate = new Date(startDateStr.split(".").reverse().join("-"));

const endDate = new Date(endDateStr.split(".").reverse().join("-"));

if (isNaN(startDate.getTime()) || isNaN(endDate.getTime()) || startDate >= endDate) {

alert("Некорректный диапазон дат.");

return false;

}

} else if (dataType === "time") {

const timeRegex = /^\d{2}:\d{2}:\d{2}-\d{2}:\d{2}:\d{2}$/;

if (!timeRegex.test(dataRange)) {

alert("Некорректный формат времени. Используйте формат ЧЧ:ММ:СС-ЧЧ:ММ:СС.");

return false;

}

const [startTime, endTime] = dataRange.split("-");

const startParts = startTime.split(":").map(Number);

const endParts = endTime.split(":").map(Number);

if (startParts.length !== 3 || endParts.length !== 3) {

alert("Некорректный формат времени. Используйте формат ЧЧ:ММ:СС-ЧЧ:ММ:СС.");

return false;

}

const startTotalSeconds = startParts[0] \* 3600 + startParts[1] \* 60 + startParts[2];

const endTotalSeconds = endParts[0] \* 3600 + endParts[1] \* 60 + endParts[2];

if (startTotalSeconds >= endTotalSeconds) {

alert("Некорректный диапазон времени.");

return false;

}

}

}

return true;

}

/\*\*

\* Отображает CSV данные в таблице.

\* @param {string} csvContent - Содержимое CSV.

\*/

function displayCSVData(csvContent) {

const csvLines = csvContent.split("\n");

const table = document.createElement("table");

const tableBody = document.createElement("tbody");

tableBody.innerHTML = "";

const headers = csvLines[0].split(",");

const headerRow = document.createElement("tr");

headers.forEach((header) => {

const th = document.createElement("th");

th.textContent = header || "null";

headerRow.appendChild(th);

});

tableBody.appendChild(headerRow);

const rowsPerPage = 30;

const totalPages = Math.ceil((csvLines.length - 1) / rowsPerPage); // Исключаем шапку

for (let i = 1; i <= rowsPerPage && i < csvLines.length; i++) { // Начинаем с 1, чтобы пропустить шапку

const row = document.createElement("tr");

const cells = csvLines[i].split(",");

console.log(`Строка ${i}:`, cells);

cells.forEach((cell, index) => {

const td = document.createElement("td");

const trimmedCell = cell.trim().replace(/"/g, "");

console.log(`Ячейка ${index} перед обрезкой:`, cell); // Отладочное сообщение

console.log(`Ячейка ${index} после обрезки:`, trimmedCell); // Отладочное сообщение

// Проверяем, является ли значение пустым (пустая строка, null, undefined)

if (trimmedCell === '' || trimmedCell === 'undefined' || trimmedCell === 'null' || trimmedCell === 'NaN') {

td.textContent = "NaN";

} else {

td.textContent = trimmedCell;

}

// Дополнительная проверка для значений 0 и false

if (trimmedCell === '0' || trimmedCell === 'false') {

console.log(`Ячейка ${index} равна 0 или false:`, trimmedCell);

}

row.appendChild(td);

});

tableBody.appendChild(row);

}

table.appendChild(tableBody);

const displayTableElement = document.getElementById("displaytable");

if (displayTableElement) {

displayTableElement.innerHTML = "";

displayTableElement.appendChild(table);

displayTableElement.style.display = "block";

} else {

console.error("Элемент с id 'displaytable' не найден");

}

const paginationContainer = document.getElementById("pagination\_\_container");

if (paginationContainer) {

paginationContainer.innerHTML = "";

const totalPagesInfo = document.createElement("div");

totalPagesInfo.className = "pagination\_\_text";

totalPagesInfo.textContent = `Всего страниц: ${totalPages}`;

paginationContainer.appendChild(totalPagesInfo);

const prevButton = document.createElement("button");

prevButton.textContent = "<";

prevButton.className = "pagination\_\_button";

prevButton.addEventListener("click", () => {

const currentPage = parseInt(paginationContainer.getAttribute("data-current-page"), 10);

if (currentPage > 1) {

displayPage(csvLines, currentPage - 1, rowsPerPage);

}

});

paginationContainer.appendChild(prevButton);

const pageButtonsContainer = document.createElement("div");

pageButtonsContainer.className = "page-buttons-container";

const currentPage = parseInt(paginationContainer.getAttribute("data-current-page"), 10) || 1;

const startPage = Math.max(1, Math.floor((currentPage - 1) / 10) \* 10 + 1);

const endPage = Math.min(totalPages, startPage + 9);

for (let i = startPage; i <= endPage; i++) {

const button = document.createElement("button");

button.textContent = i;

button.className = "pagination\_\_button";

button.addEventListener("click", () => {

displayPage(csvLines, i, rowsPerPage);

});

pageButtonsContainer.appendChild(button);

}

paginationContainer.appendChild(pageButtonsContainer);

const nextButton = document.createElement("button");

nextButton.textContent = ">";

nextButton.className = "pagination\_\_button";

nextButton.addEventListener("click", () => {

const currentPage = parseInt(paginationContainer.getAttribute("data-current-page"), 10);

if (currentPage < totalPages) {

displayPage(csvLines, currentPage + 1, rowsPerPage);

}

});

paginationContainer.appendChild(nextButton);

paginationContainer.setAttribute("data-current-page", "1");

} else {

console.error("Элемент с id 'pagination\_\_container' не найден");

}

}

/\*\*

\* Отображает определенную страницу CSV данных.

\* @param {Array} csvLines - Строки CSV.

\* @param {number} page - Номер страницы.

\* @param {number} rowsPerPage - Количество строк на странице.

\*/

function displayPage(csvLines, page, rowsPerPage) {

const startIndex = (page - 1) \* rowsPerPage + 1;

const endIndex = Math.min(startIndex + rowsPerPage, csvLines.length);

const tableBody = document.createElement("tbody");

for (let i = startIndex; i < endIndex; i++) {

const row = document.createElement("tr");

const cells = csvLines[i].split(",");

cells.forEach((cell, index) => {

const td = document.createElement("td");

const trimmedCell = cell.trim().replace(/"/g, "");

if (trimmedCell === '' || trimmedCell === 'undefined' || trimmedCell === 'null') {

td.textContent = "NaN";

} else {

td.textContent = trimmedCell;

}

row.appendChild(td);

});

tableBody.appendChild(row);

}

const table = document.createElement("table");

table.appendChild(tableBody);

const displayTableElement = document.getElementById("displaytable");

if (displayTableElement) {

displayTableElement.innerHTML = "";

displayTableElement.appendChild(table);

} else {

console.error("Элемент с id 'displaytable' не найден");

}

const paginationContainer = document.getElementById("pagination\_\_container");

if (paginationContainer) {

paginationContainer.setAttribute("data-current-page", page);

} else {

console.error("Элемент с id 'pagination\_\_container' не найден");

}

}

/\*\*

\* Очищает таблицу и сбрасывает форму.

\*/

function clearTable() {

const displaytable = document.getElementById("displaytable");

if (!displaytable) {

console.error('Элемент с ID "displaytable" не найден');

return;

}

displaytable.innerHTML = "";

displaytable.style.display = "none";

clearIndexedDB();

localStorage.removeItem("csvData");

const paginationContainer = document.getElementById("pagination\_\_container");

if (paginationContainer) {

paginationContainer.innerHTML = "";

}

document.getElementById("num\_rows").value = "";

document.getElementById("filename").value = "";

document.getElementById("indexType").value = "none";

const columnsContainer = document.getElementById("columns");

columnsContainer.innerHTML = "";

columnCount = 0;

// Очистка и скрытие поля для ввода пользовательского диапазона индексов

const customIndexRange = document.getElementById("customIndexRange");

customIndexRange.value = "";

customIndexRange.style.display = "none";

}

/\*\*

\* Сохраняет CSV данные в файл.

\*/

async function saveCSV() {

let csvContent = localStorage.getItem("csvData");

if (!csvContent) {

// Если данных нет в localStorage, загружаем их из IndexedDB

csvContent = await loadModelFromIndexedDB();

if (!csvContent) {

alert("Сначала создайте или загрузите CSV данные.");

return;

}

}

const processMessage = document.getElementById("processMessage");

const messageBox = processMessage.querySelector(".message-box");

// Устанавливаем текст сообщения

messageBox.textContent = "Сохраняем данные... Пожалуйста, подождите.";

// Отображаем контейнер

processMessage.style.display = "flex";

const filename = document.getElementById("filename").value + ".csv";

const bom = "\uFEFF";

const blob = new Blob([bom + csvContent], {

type: "text/csv;charset=utf-8;"

});

const url = URL.createObjectURL(blob);

const a = document.createElement("a");

a.href = url;

a.download = filename;

a.click();

URL.revokeObjectURL(url);

console.log("Сохранено CSV содержимое:", csvContent);

processMessage.style.display = "none"; // Скрываем сообщение после завершения сохранения

}

/\*\*

\* Обрабатывает загрузку CSV файла.

\* @param {Event} event - Событие загрузки файла.

\*/

async function handleCSVFile(event) {

const file = event.target.files[0];

const errorMessages = document.getElementById("errorMessages");

errorMessages.innerHTML = "";

if (file) {

const processMessage = document.getElementById("processMessage");

const messageBox = processMessage.querySelector(".message-box");

// Устанавливаем текст сообщения

messageBox.textContent = "Загружаем данные... Пожалуйста, подождите.";

// Отображаем контейнер

processMessage.style.display = "flex";

Papa.parse(file, {

header: true,

dynamicTyping: false,

skipEmptyLines: false, // Не пропускать пустые строки

complete: async function (results) {

csvData = results.data;

let columns = Object.keys(csvData[0]);

// Проверка и присвоение имен пустым заголовкам

columns = columns.map(

(column, index) => column.trim() || `Column\_${index + 1}`

);

document.getElementById("columns").innerHTML = "";

columnCount = 0;

columns.forEach((columnName) => {

const dataType =

columnMetadata[columnName]?.dataType || "fromloadcsv";

const dataRange = csvData

.map((row) => row[columnName])

.filter((value, index, self) => self.indexOf(value) === index)

.join(", ");

addColumn(columnName, dataType, dataRange);

// Обновление поля ввода диапазона

const uniqueValues = [

...new Set(csvData.map((row) => row[columnName]))

];

document.getElementById(`data\_range\_${columnCount}`).value =

uniqueValues.join(", ");

});

// Удаляем кавычки из данных

csvData.forEach((row) => {

for (let key in row) {

if (typeof row[key] === "string") {

row[key] = row[key].replace(/"/g, "");

}

}

});

// Обработка пустых значений

csvData.forEach((row) => {

for (let key in row) {

if (row[key] === "") {

row[key] = ""; // Замена пустых значений на пустую строку

}

}

});

// Удаление пустых строк

csvData = csvData.filter((row) =>

Object.values(row).some(

(value) => value !== null && value !== undefined && value !== ""

)

);

// Проверка на наличие ошибок

let hasErrors = false;

let errorReport = "Обнаружены следующие ошибки в файле:\n";

csvData.forEach((row, rowIndex) => {

let rowErrors = [];

Object.keys(row).forEach((key) => {

if (

row[key] === "" ||

row[key] === null ||

row[key] === undefined

) {

rowErrors.push(`Пустое значение в столбце "${key}"`);

}

});

if (rowErrors.length > 0) {

hasErrors = true;

errorReport += `Строка ${rowIndex + 1}: ${rowErrors.join(", ")}\n`;

}

});

if (hasErrors) {

alert(errorReport);

}

const csvContent = Papa.unparse(csvData);

const maxLocalStorageSize = 5 \* 1024 \* 1024; // 5 MB

if (csvContent.length < maxLocalStorageSize) {

// Если данные меньше 5 MB, сохраняем в localStorage

localStorage.setItem("csvData", csvContent);

alert("CSV данные сохранены в localStorage");

} else {

// Если данные больше 5 MB, сохраняем в IndexedDB частями

await saveModelToIndexedDBInParts(csvContent);

alert("CSV данные сохранены в IndexedDB");

}

displayCSVData(csvContent);

isCSVLoaded = true;

// Обновление полей формы

document.getElementById("num\_rows").value = csvData.length;

document.getElementById("filename").value = file.name.replace(

".csv",

""

);

updateHintSection(file.name);

console.log("Загружен CSV Data:", csvData);

// Аналитика

const totalRowsdata = csvData.length;

const totalColumns = Object.keys(csvData[0]).length;

let emptyRows = 0;

let missingValues = 0;

let nullValues = 0;

let undefinedValues = 0;

let nanValues = 0;

csvData.forEach((row) => {

if (Object.values(row).every((value) => value === "")) {

emptyRows++;

}

Object.values(row).forEach((value) => {

if (value === null) nullValues++;

if (value === undefined) undefinedValues++;

if (value === "" || value === "NaN") nanValues++;

if (

value === "" ||

value === null ||

value === undefined ||

value === "NaN"

) {

missingValues++;

}

});

});

const analyticsContent = `

<p>Общее количество строк: ${totalRowsdata}</p>

<p>Общее количество столбцов: ${totalColumns}</p>

<p>Пустые строки: ${emptyRows}</p>

<p>Пропущенные значения: ${missingValues}</p>

<p>NULL значения: ${nullValues}</p>

<p>Неопределенные значения: ${undefinedValues}</p>

<p>NaN-значения: ${nanValues}</p>

`;

document.getElementById("analyticsContent").innerHTML =

analyticsContent;

processMessage.style.display = "none"; // Скрываем сообщение в случае ошибки

},

error: function (error, file) {

errorMessages.innerHTML = `Ошибка парсинга: ${error.message}`;

console.error("Ошибка парсинга:", error);

}

});

}

}

/\*\*

\* Обновляет секцию подсказок для Jupyter Notebook и Google Colab.

\* @param {string} filename - Имя файла.

\*/

function updateHintSection(filename) {

const hintCode = `

Основные функции (пример)

import pandas as pd

import io

# Загрузка данных

data = pd.read\_csv(io.BytesIO(uploaded['${filename}.csv']))

# Получаем размеры данных (количество строк и столбцов)

data.shape

# Выводим информацию о данных, включая типы данных и количество непустых значений

data.info()

# Получаем список всех столбцов в данных

data.columns

# Выводим статистическое описание данных, включая среднее значение, стандартное отклонение и т.д.

data.describe()

df.describe(include='all')

# Просмотр первых строк данных

print(data.head())

# Проверка на наличие пропусков

print("Проверка на наличие пропусков:")

print(data.isnull().sum())

# Удаление дубликатов (если есть)

duplicate\_count = data.duplicated().sum()

if duplicate\_count > 0:

data = data.drop\_duplicates()

print(f"Удалено {duplicate\_count} дубликатов")

# Подсчет уникальных значений в каждом столбце

unique\_counts = data.nunique()

print("Уникальные значения в каждом столбце:")

print(unique\_counts)

# Статистическая сводка

print("Статистическая сводка:")

print(data.describe(include='all'))

# Корреляционная матрица (если есть числовые данные)

if data.select\_dtypes(include=['number']).shape[1] > 0:

correlation\_matrix = data.corr()

print("Корреляционная матрица:")

print(correlation\_matrix)

# Визуализация данных

import matplotlib.pyplot as plt

import seaborn as sns

# Гистограммы для числовых данных

data.select\_dtypes(include=['number']).hist(bins=30, figsize=(20, 15))

plt.show()

# Тепловая карта корреляций

if data.select\_dtypes(include=['number']).shape[1] > 0:

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

plt.show()

# Распределение категориальных данных

for column in data.select\_dtypes(include=['object']).columns:

plt.figure(figsize=(10, 5))

sns.countplot(y=column, data=data)

plt.title(f'Распределение {column}')

plt.show()

`;

document.getElementById("hintCode").textContent = hintCode;

}

/\*\*

\* Анализирует данные в указанном столбце CSV.

\* @param {string} columnName - Название столбца для анализа.

\* @returns {Promise<Object>} - Возвращает промис, который разрешается объектом с метриками анализа столбца.

\*/

async function analyzeColumnData(columnName) {

let csvContent = localStorage.getItem("csvData");

if (!csvContent) {

// Если данных нет в localStorage, загружаем их из IndexedDB

csvContent = await loadModelFromIndexedDB();

if (!csvContent) {

alert("Сначала создайте или загрузите CSV данные.");

return {

uniqueCount: 0,

duplicateCount: 0,

uniqueValues: "Нет уникальных значений",

duplicateValues: "Нет повторяющихся значений",

missingValues: 0,

nullValues: 0,

undefinedValues: 0,

nanValues: 0

};

}

}

const csvLines = csvContent.split("\n");

const headers = csvLines[0].split(",");

const data = csvLines.slice(1).map(line => line.split(","));

const columnIndex = headers.indexOf(columnName);

if (columnIndex === -1) {

console.error(`Столбец ${columnName} не найден в данных.`);

return {

uniqueCount: 0,

duplicateCount: 0,

uniqueValues: "Нет уникальных значений",

duplicateValues: "Нет повторяющихся значений",

missingValues: 0,

nullValues: 0,

undefinedValues: 0,

nanValues: 0

};

}

const columnData = data.map(row => row[columnIndex] !== undefined ? row[columnIndex] : "");

const valueCounts = columnData.reduce((acc, value) => {

if (value !== "" && value !== "NaN") {

acc[value] = (acc[value] || 0) + 1;

}

return acc;

}, {});

const uniqueValues = Object.keys(valueCounts).filter(value => valueCounts[value] === 1);

const duplicateValues = Object.keys(valueCounts).filter(value => valueCounts[value] > 1);

let missingValues = 0;

let nullValues = 0;

let undefinedValues = 0;

let nanValues = 0;

columnData.forEach(value => {

console.log(`Обрабатываемое значение: ${value}`);

if (value === null) {

nullValues++;

console.log(`nullValues увеличено: ${nullValues}`);

} else if (value === undefined) {

undefinedValues++;

console.log(`undefinedValues увеличено: ${undefinedValues}`);

} else if (value === '' || value === 'NaN') {

nanValues++;

console.log(`nanValues увеличено: ${nanValues}`);

}

if (value === '' || value === null || value === undefined || value === 'NaN') {

missingValues++;

console.log(`missingValues увеличено: ${missingValues}`);

}

});

console.log(`Анализируемый столбец: ${columnName}`);

console.log("Данные столбца:", columnData);

console.log("Количество значений:", valueCounts);

console.log("Уникальные значения:", uniqueValues);

console.log("Повторяющиеся значения:", duplicateValues);

console.log("Пропущенные значения:", missingValues);

console.log("NULL значения:", nullValues);

console.log("Неопределенные значения:", undefinedValues);

console.log("NaN-значения:", nanValues);

return {

uniqueCount: uniqueValues.length,

duplicateCount: duplicateValues.length,

uniqueValues: uniqueValues.join(", ") || "Нет уникальных значений",

duplicateValues: duplicateValues.join(", ") || "Нет повторяющихся значений",

missingValues: missingValues,

nullValues: nullValues,

undefinedValues: undefinedValues,

nanValues: nanValues

};

}

/\*\*

\* Проверяет и запрашивает ввод данных перед генерацией CSV.

\* @returns {boolean} - Результат проверки.

\*/

function checkAndPrompt() {

const numRows = parseInt(document.getElementById("num\_rows").value, 10);

const filename = document.getElementById("filename").value;

const indexType = document.getElementById("indexType").value;

if (isNaN(numRows) || numRows <= 0) {

alert("Количество строк должно быть положительным числом.");

return false;

}

if (!filename) {

alert("Имя файла не может быть пустым.");

return false;

}

if (indexType === "none" && columnCount === 0) {

alert("Введите больше данных. Нажмите кнопку Добавить столбец");

return false;

}

const columnNames = [];

for (let i = 1; i <= columnCount; i++) {

const columnName = document.getElementById(`column\_name\_${i}`).value;

if (columnNames.includes(columnName)) {

const changeData = confirm(`"${columnName}" уже существует`);

if (changeData) {

return false;

}

}

columnNames.push(columnName);

}

return true;

}

/\*\*

\* Создает и обучает модель машинного обучения.

\* @param {Array} inputShape - Форма входных данных.

\* @param {tf.Tensor} xTrain - Тензор обучающих данных.

\* @param {tf.Tensor} yTrain - Тензор целевых данных.

\* @returns {Promise<void>} - Промис, который разрешается при успешном создании и обучении модели.

\*/

async function createAndTrainModel(inputShape, xTrain, yTrain) {

try {

if (yTrain.shape[1] === 0) {

throw new Error('yTrain.shape[1] не может быть равно 0');

}

// Определяем архитектуру модели

model = tf.sequential();

model.add(tf.layers.dense({ units: 128, activation: 'relu', inputShape: inputShape }));

model.add(tf.layers.dense({ units: yTrain.shape[1], activation: 'linear' }));

// Компилируем модель

model.compile({

optimizer: 'adam',

loss: 'meanSquaredError',

metrics: ['mae']

});

// Обучаем модель

await model.fit(xTrain, yTrain, {

epochs: 10,

batchSize: 32,

callbacks: {

onEpochEnd: (epoch, logs) => console.log(`Epoch ${epoch + 1}: loss = ${logs.loss}`)

}

});

console.log('Модель создана и обучена успешно');

} catch (error) {

console.error('Ошибка создания и обучения модели:', error);

}

}

/\*\*

\* Анализирует частотность слов в текстовых данных.

\* @param {string} columnName - Название столбца.

\* @returns {Object} - Объект с метриками анализа частотности слов.

\*/

async function analyzeWordFrequency(columnName) {

let csvContent = localStorage.getItem("csvData");

if (!csvContent) {

// Если данных нет в localStorage, загружаем их из IndexedDB

csvContent = await loadModelFromIndexedDB();

if (!csvContent) {

alert("Сначала создайте или загрузите CSV данные.");

return {

wordFrequency: {},

totalWords: 0,

uniqueWords: 0

};

}

}

const csvLines = csvContent.split("\n");

const headers = csvLines[0].split(",");

const data = csvLines.slice(1).map(line => line.split(","));

const columnIndex = headers.indexOf(columnName);

if (columnIndex === -1) {

console.error(`Столбец ${columnName} не найден в данных.`);

return {

wordFrequency: {},

totalWords: 0,

uniqueWords: 0

};

}

const columnData = data.map(row => row[columnIndex] !== undefined ? row[columnIndex] : "").join(" ").toLowerCase();

// Use a regular expression to match words and decimal numbers

const words = columnData.match(/\b\d+(\.\d+)?\b|\b\w+\b/g);

const wordFrequency = words.reduce((acc, word) => {

acc[word] = (acc[word] || 0) + 1;

return acc;

}, {});

const totalWords = words.length;

const uniqueWords = Object.keys(wordFrequency).length;

console.log(`Анализ частотности значений в столбце ${columnName}`);

console.log("Частотность цифр и слов:", wordFrequency);

console.log("Общее количество данных:", totalWords);

console.log("Количество различных используемых цифр и слов:", uniqueWords);

return {

wordFrequency: wordFrequency,

totalWords: totalWords,

uniqueWords: uniqueWords

};

}

/\*\*

\* Построение тепловой карты корреляции.

\* @param {Array} data - Данные для построения тепловой карты.

\* @param {Array} headers - Заголовки столбцов.

\*/

function buildHeatmap(data, headers) {

// Фильтрация столбцов, содержащих только числовые значения

const numericColumns = headers.filter(header => {

return data.every(row => !isNaN(parseFloat(row[headers.indexOf(header)])));

});

const numericData = numericColumns.map(column => {

return data.map(row => parseFloat(row[headers.indexOf(column)]));

});

const correlationMatrix = [];

const numCols = numericColumns.length;

for (let i = 0; i < numCols; i++) {

correlationMatrix[i] = [];

for (let j = 0; j < numCols; j++) {

const col1 = numericData[i];

const col2 = numericData[j];

const correlation = calculateCorrelation(col1, col2);

correlationMatrix[i][j] = correlation;

}

}

const heatmapData = {

z: correlationMatrix,

x: numericColumns,

y: numericColumns,

type: 'heatmap',

colorscale: 'Viridis'

};

const layout = {

title: 'Тепловая карта корреляции столбцов с числовыми значениями',

xaxis: { tickangle: -45 },

yaxis: { tickangle: -45 },

annotations: []

};

Plotly.newPlot('heatmap', [heatmapData], layout);

}

/\*\*

\* Вычисляет корреляцию между двумя столбцами данных.

\* @param {Array} col1 - Данные первого столбца.

\* @param {Array} col2 - Данные второго столбца.

\* @returns {number} - Корреляция между столбцами.

\*/

function calculateCorrelation(col1, col2) {

const mean1 = col1.reduce((sum, val) => sum + val, 0) / col1.length;

const mean2 = col2.reduce((sum, val) => sum + val, 0) / col2.length;

const numerator = col1.map((val, i) => (val - mean1) \* (col2[i] - mean2)).reduce((sum, val) => sum + val, 0);

const denominator1 = col1.map(val => Math.pow(val - mean1, 2)).reduce((sum, val) => sum + val, 0);

const denominator2 = col2.map(val => Math.pow(val - mean2, 2)).reduce((sum, val) => sum + val, 0);

const correlation = numerator / Math.sqrt(denominator1 \* denominator2);

return correlation;

}

/\*\*

\* Отображает аналитику данных.

\*/

async function showAnalytics() {

try {

console.log("Начало процесса аналитики...");

const loadingContainer = document.getElementById("loadingContainer");

let csvContent = localStorage.getItem("csvData");

if (!csvContent) {

// Если данных нет в localStorage, загружаем их из IndexedDB

csvContent = await loadModelFromIndexedDB();

if (!csvContent) {

alert("Сначала создайте или загрузите CSV данные.");

loadingContainer.style.display = "none"; // Скрываем контейнер загрузки

return;

}

}

if (!loadingContainer) {

console.error("Элемент loadingContainer не найден в документе.");

return;

}

const messageBox = loadingContainer.querySelector(".message-box");

messageBox.textContent = "Анализируем данные... Пожалуйста, подождите.";

loadingContainer.style.display = "flex"; // Отображаем контейнер загрузки

const parsedData = Papa.parse(csvContent, { header: true });

// Сохраняем исходные данные

let originalData = parsedData.data;

const headersIndexedDB = parsedData.meta.fields;

let metricsHeaders = headersIndexedDB.filter(header => !isNaN(Number(parsedData.data[0][header])));

let dataDb = parsedData.data.map(row => metricsHeaders.map(header => {

const value = Number(row[header]);

return isNaN(value) ? null : value;

}));

const chunkSize = 100000; // Размер одной части

const totalRows = dataDb.length;

// Заполняем пропущенные значения средним значением соответствующих столбцов

const numCols = dataDb[0].length;

let means = Array(numCols).fill(0);

let count = Array(numCols).fill(0);

dataDb.forEach(row => {

row.forEach((value, colIndex) => {

if (value !== null) {

means[colIndex] += value;

count[colIndex]++;

}

});

});

for (let i = 0; i < totalRows; i += chunkSize) {

const chunk = dataDb.slice(i, i + chunkSize);

await analyzeChunk(chunk, headersIndexedDB);

}

// Обновляем интерфейс с результатами анализа

// updateAnalyticsUI(headersIndexedDB, dataDb);

means = means.map((mean, index) => count[index] > 0 ? mean / count[index] : 0);

dataDb = dataDb.map(row => row.map((value, colIndex) => value !== null ? value : means[colIndex]));

console.log('Обработанные данные:', dataDb);

const numRows = dataDb.length;

const tensorData = tf.tensor2d(dataDb, [numRows, numCols]);

console.log('Тензор данных:', tensorData);

const mean = tensorData.mean(0, true);

const std = tensorData.sub(mean).square().mean(0, true).sqrt();

const normalizedData = tensorData.sub(mean).div(std);

console.log('Нормализованные данные:', normalizedData);

const splitIndex = Math.floor(normalizedData.shape[0] \* 0.8);

const xTrain = normalizedData.slice([0, 0], [splitIndex, normalizedData.shape[1]]);

const yTrain = normalizedData.slice([0, 0], [splitIndex, normalizedData.shape[1]]);

const xTest = normalizedData.slice([splitIndex, 0], [-1, normalizedData.shape[1]]);

console.log('Обучающие данные:', xTrain, yTrain);

console.log('Тестовые данные:', xTest);

if (yTrain.shape[1] === 0) {

console.error('yTrain.shape[1] не может быть равно 0');

return;

}

const inputShape = [xTrain.shape[1]];

await createAndTrainModel(inputShape, xTrain, yTrain);

const predictions = model.predict(xTest);

console.log('Прогнозы:', predictions);

const predictionsDenormalized = predictions.mul(std).add(mean);

console.log('Денормализованные прогнозы:', predictionsDenormalized);

const meanPredictions = predictionsDenormalized.mean(0);

console.log('Усредненный прогноз:', meanPredictions);

const resultsDiv = document.getElementById('results');

resultsDiv.innerHTML = '';

const table = document.createElement("table");

const thead = document.createElement("thead");

const tbody = document.createElement("tbody");

const headerRow = document.createElement('tr');

metricsHeaders.forEach(header => {

const th = document.createElement('th');

th.textContent = header;

headerRow.appendChild(th);

});

thead.appendChild(headerRow);

const meanPredictionsArray = meanPredictions.arraySync();

const tr = document.createElement('tr');

meanPredictionsArray.forEach(value => {

const td = document.createElement('td');

td.textContent = value.toFixed(2);

tr.appendChild(td);

});

tbody.appendChild(tr);

table.appendChild(thead);

table.appendChild(tbody);

resultsDiv.appendChild(table);

const csvData = parsedData.data;

const columns = Object.keys(csvData[0]);

const filteredColumns = columns.filter(column => {

return !isNaN(Number(csvData[0][column]));

});

const traces = filteredColumns.map(column => {

return {

x: csvData.map((row, index) => index),

y: csvData.map(row => row[column]),

mode: 'lines+markers',

name: column

};

});

const layoutgraph = {

autosize: true,

width: '100%',

height: 400,

margin: {

l: 50,

r: 50,

b: 100,

t: 100,

pad: 4

},

title: 'График данных',

barmode: 'group' // Группировка столбцов по категориям

};

Plotly.newPlot('graph', traces, layoutgraph);

const chartDiv = document.getElementById('chart');

chartDiv.innerHTML = '';

const trace = {

x: metricsHeaders,

y: meanPredictionsArray,

type: 'bar',

marker: {

color: 'rgba(25, 103, 210, 0.8)',

line: {

color: 'rgba(25, 103, 210, 1);',

width: 1.5

}

}

};

const layout = {

title: 'Усредненные прогнозы',

autosize: true,

width: '100%',

height: 400,

margin: {

l: 50,

r: 50,

b: 100,

t: 100,

pad: 4

}

};

Plotly.newPlot(chartDiv, [trace], layout);

const csvLines = csvContent.split("\n");

const headers = csvLines[0].split(",");

const data = csvLines.slice(1).map(line => line.split(","));

const totalRowsdata = data.length;

for (let i = 0; i <totalRowsdata; i += chunkSize) {

const chunk = data.slice(i, i + chunkSize);

await analyzeChunk(chunk, headers);

}

// Обновляем интерфейс с результатами анализа

// updateAnalyticsUI(headers, data);

const analyticsContent = document.getElementById("analyticsContent");

analyticsContent.innerHTML = "";

// Добавляем метрики для аналитики в начало

const totalColumns = headers.length;

let emptyRows = 0;

let missingValues = 0;

let nullValues = 0;

let undefinedValues = 0;

let nanValues = 0;

data.forEach(row => {

if (Object.values(row).every(value => value === "")) {

emptyRows++;

}

Object.values(row).forEach(value => {

if (value === null) {

nullValues++;

} else if (value === undefined) {

undefinedValues++;

} else if (value === '' || value === 'NaN') {

nanValues++;

}

if (value === '' || value === null || value === undefined || value === 'NaN') {

missingValues++;

}

});

});

const analyticsMetrics = document.createElement("div");

analyticsMetrics.innerHTML = `

<p>Всего строк: ${totalRowsdata}</p>

<p>Всего столбцов: ${totalColumns}</p>

<p>Пустые строки: ${emptyRows}</p>

<p>Пропущенные значения: ${missingValues}</p>

<p>NULL значения: ${nullValues}</p>

<p>Неопределенные значения: ${undefinedValues}</p>

<p>NaN-значения: ${nanValues}</p>

`;

analyticsContent.appendChild(analyticsMetrics);

const analyticsTable = document.createElement("table");

const analyticsThead = document.createElement("thead");

const analyticsTbody = document.createElement("tbody");

// Создаем заголовок таблицы с метриками

const analyticsMetricsRow = document.createElement("tr");

const analyticsMetricsHeaders = [

"Имя столбца",

"Количество уникальных значений",

"Уникальные значения",

"Количество повторяющихся значений",

"Повторяющиеся значения",

"Пропущенные значения",

"NULL значения",

"Неопределенные значения",

"NaN-значения",

"Частотность слов",

"Общее количество данных",

"Количество различных используемых цифр и слов"

];

analyticsMetricsHeaders.forEach(header => {

const th = document.createElement("th");

th.textContent = header;

th.style.verticalAlign = "top"; // Выравнивание текста по верхней границе

analyticsMetricsRow.appendChild(th);

});

analyticsThead.appendChild(analyticsMetricsRow);

// Создаем строки для каждого столбца

for (let i = 0; i < headers.length; i++) {

const header = headers[i];

try {

const analysisResult = await analyzeColumnData(header);

const wordFrequencyResult = await analyzeWordFrequency(header);

console.log(`Результат анализа для столбца ${header}:`, analysisResult);

console.log(

`Результат анализа частотности слов для столбца ${header}:`,

wordFrequencyResult

);

const row = document.createElement("tr");

const th = document.createElement("th");

th.textContent = header;

th.style.verticalAlign = "top"; // Выравнивание текста по верхней границе

row.appendChild(th);

const uniqueCountCell = document.createElement("td");

uniqueCountCell.textContent = analysisResult.uniqueCount;

uniqueCountCell.style.backgroundColor = "#f9f9f9"; // Цвет фона ячейки

uniqueCountCell.style.verticalAlign = "top"; // Выравнивание текста по верхней границе

row.appendChild(uniqueCountCell);

const uniqueValuesCell = document.createElement("td");

uniqueValuesCell.textContent = analysisResult.uniqueValues;

uniqueValuesCell.style.backgroundColor = "#f9f9f9"; // Цвет фона ячейки

uniqueValuesCell.style.verticalAlign = "top"; // Выравнивание текста по верхней границе

row.appendChild(uniqueValuesCell);

const duplicateCountCell = document.createElement("td");

duplicateCountCell.textContent = analysisResult.duplicateCount;

duplicateCountCell.style.backgroundColor = "#f9f9f9"; // Цвет фона ячейки

duplicateCountCell.style.verticalAlign = "top"; // Выравнивание текста по верхней границе

row.appendChild(duplicateCountCell);

const duplicateValuesCell = document.createElement("td");

duplicateValuesCell.textContent = analysisResult.duplicateValues;

duplicateValuesCell.style.backgroundColor = "#f9f9f9"; // Цвет фона ячейки

duplicateValuesCell.style.verticalAlign = "top"; // Выравнивание текста по верхней границе

row.appendChild(duplicateValuesCell);

const missingValuesCell = document.createElement("td");

missingValuesCell.textContent = analysisResult.missingValues;

missingValuesCell.style.backgroundColor = "#f9f9f9"; // Цвет фона ячейки

missingValuesCell.style.verticalAlign = "top"; // Выравнивание текста по верхней границе

row.appendChild(missingValuesCell);

const nullValuesCell = document.createElement("td");

nullValuesCell.textContent = analysisResult.nullValues;

nullValuesCell.style.backgroundColor = "#f9f9f9"; // Цвет фона ячейки

nullValuesCell.style.verticalAlign = "top"; // Выравнивание текста по верхней границе

row.appendChild(nullValuesCell);

const undefinedValuesCell = document.createElement("td");

undefinedValuesCell.textContent = analysisResult.undefinedValues;

undefinedValuesCell.style.backgroundColor = "#f9f9f9"; // Цвет фона ячейки

undefinedValuesCell.style.verticalAlign = "top"; // Выравнивание текста по верхней границе

row.appendChild(undefinedValuesCell);

const nanValuesCell = document.createElement("td");

nanValuesCell.textContent = analysisResult.nanValues;

nanValuesCell.style.backgroundColor = "#f9f9f9"; // Цвет фона ячейки

nanValuesCell.style.verticalAlign = "top"; // Выравнивание текста по верхней границе

row.appendChild(nanValuesCell);

const wordFrequencyCell = document.createElement("td");

wordFrequencyCell.innerHTML = JSON.stringify(wordFrequencyResult.wordFrequency, null, 2);

wordFrequencyCell.style.backgroundColor = "#f9f9f9"; // Цвет фона ячейки

wordFrequencyCell.style.verticalAlign = "top"; // Выравнивание текста по верхней границе

row.appendChild(wordFrequencyCell);

const totalWordsCell = document.createElement("td");

totalWordsCell.textContent = wordFrequencyResult.totalWords;

totalWordsCell.style.backgroundColor = "#f9f9f9"; // Цвет фона ячейки

totalWordsCell.style.verticalAlign = "top"; // Выравнивание текста по верхней границе

row.appendChild(totalWordsCell);

const uniqueWordsCell = document.createElement("td");

uniqueWordsCell.textContent = wordFrequencyResult.uniqueWords;

uniqueWordsCell.style.backgroundColor = "#f9f9f9"; // Цвет фона ячейки

uniqueWordsCell.style.verticalAlign = "top"; // Выравнивание текста по верхней границе

row.appendChild(uniqueWordsCell);

analyticsTbody.appendChild(row);

} catch (error) {

console.error(`Ошибка при анализе столбца ${header}:`, error);

}

}

analyticsTable.appendChild(analyticsThead);

analyticsTable.appendChild(analyticsTbody);

analyticsTable.className = "analytics-table";

analyticsContent.appendChild(analyticsTable);

document.getElementById("analyticsModal").style.display = "block";

const chartContainer = document.getElementById("chartContainer");

chartContainer.innerHTML = "";

// Контейнер для строк с NaN значениями

const nanRowsContainer = document.getElementById("nanRowsContainer");

nanRowsContainer.innerHTML = "";

headers.forEach((header, index) => {

const columnData = data.map(row => row[index] !== undefined ? row[index] : "NULL");

const uniqueValues = [...new Set(columnData)];

// Сортировка данных по количеству значений

let sortedValues = uniqueValues.map(value => ({

value: value,

count: columnData.filter(v => v === value).length

})).sort((a, b) => a.count - b.count);

const valueCounts = sortedValues.map(item => ({

value: item.value,

count: item.count

}));

const canvas = document.createElement("canvas");

canvas.id = `chart\_${index}`;

chartContainer.appendChild(canvas);

const ctx = canvas.getContext('2d');

new Chart(ctx, {

type: 'bar',

data: {

labels: valueCounts.map(vc => vc.value),

datasets: [{

label: `Количество ${header}`,

data: valueCounts.map(vc => vc.count),

backgroundColor: 'rgba(25, 103, 210, 0.8)',

borderColor: 'rgba(25, 103, 210, 1);',

borderWidth: 1

}]

},

options: {

scales: {

y: {

beginAtZero: true

}

}

}

});

// Ищем минимальное и максимальное значение в числовых колонках

if (uniqueValues.every(value => !isNaN(parseFloat(value)))) {

const numericValues = columnData.map(value => parseFloat(value)).filter(value => !isNaN(value));

const minValue = Math.min(...numericValues);

const maxValue = Math.max(...numericValues);

const minRows = [];

const maxRows = [];

// Одновременно ищем строки с минимальным и максимальным значением

data.forEach((row, rowIndex) => {

const value = parseFloat(row[index]);

if (value === minValue) {

minRows.push(row);

}

if (value === maxValue) {

maxRows.push(row);

}

});

// Отображаем строки с минимальным значением

if (minRows.length > 0) {

const minRowDiv = document.createElement("div");

minRowDiv.innerHTML = `<strong>Минимальное значение для ${header}:</strong>`;

chartContainer.appendChild(minRowDiv);

const minRowsTable = createTable(headers, minRows);

chartContainer.appendChild(minRowsTable);

}

// Отображаем строки с максимальным значением

if (maxRows.length > 0) {

const maxRowDiv = document.createElement("div");

maxRowDiv.innerHTML = `<strong>Максимальное значение для ${header}:</strong>`;

chartContainer.appendChild(maxRowDiv);

const maxRowsTable = createTable(headers, maxRows);

chartContainer.appendChild(maxRowsTable);

}

}

// Отображаем строки с NaN-значениями из исходных данных

const nanRows = originalData.filter((row) =>

headers.some(

(header) =>

row[header] === "NaN" ||

row[header] === "" ||

row[header] === null ||

row[header] === undefined

)

);

const nanRowsContainer = document.getElementById("nanRowsContainer");

nanRowsContainer.innerHTML = ""; // Очищаем контейнер перед добавлением новых данных

if (nanRows.length > 0) {

const nanRowsTable = document.createElement("table");

const nanRowsThead = document.createElement("thead");

const nanRowsTbody = document.createElement("tbody");

const nanRowsHeaderRow = document.createElement("tr");

headers.forEach((h) => {

const th = document.createElement("th");

th.textContent = h;

nanRowsHeaderRow.appendChild(th);

});

nanRowsThead.appendChild(nanRowsHeaderRow);

nanRows.forEach((row) => {

const tr = document.createElement("tr");

headers.forEach((h) => {

const td = document.createElement("td");

td.textContent = row[h];

tr.appendChild(td);

});

nanRowsTbody.appendChild(tr);

});

nanRowsTable.appendChild(nanRowsThead);

nanRowsTable.appendChild(nanRowsTbody);

nanRowsContainer.appendChild(nanRowsTable);

} else {

// Создаем сообщение, если строки с NaN-значениями не найдены

const noNanRowsMessage = document.createElement("div");

noNanRowsMessage.textContent = "Строки с NaN-значениями не найдены.";

noNanRowsMessage.style.color = "red";

nanRowsContainer.appendChild(noNanRowsMessage);

}

});

console.log("Анализ данных:", data);

// Построение тепловой карты корреляции

buildHeatmap(dataDb, headersIndexedDB);

// Обновляем секцию подсказок

const filename = document.getElementById("filename").value;

updateHintSection(filename);

console.log("Аналитика завершена: скрываем loadingContainer");

loadingContainer.style.display = "none"; // Скрываем контейнер загрузки после завершения анализа

} catch (error) {

console.error("Ошибка анализа:", error);

}

}

/\*\*

\* Анализирует часть данных CSV.

\* @param {Array} chunk - Часть данных для анализа.

\* @param {Array} headers - Заголовки столбцов.

\* @returns {Promise<void>} - Возвращает промис, который разрешается после завершения анализа части данных.

\*/

async function analyzeChunk(chunk, headers) {

// Анализируем текущую часть данных

for (let i = 0; i < headers.length; i++) {

const header = headers[i];

try {

const analysisResult = await analyzeColumnData(header);

const wordFrequencyResult = await analyzeWordFrequency(header);

console.log(`Результат анализа для столбца ${header}:`, analysisResult);

console.log(`Результат анализа частотности слов для столбца ${header}:`, wordFrequencyResult);

} catch (error) {

console.error(`Ошибка при анализе столбца ${header}:`, error);

}

}

}

/\*\*

\* Создает таблицу для отображения строк.

\* @param {Array} headers - Заголовки столбцов.

\* @param {Array} rows - Строки данных.

\* @returns {HTMLTableElement} - Элемент таблицы.

\*/

function createTable(headers, rows) {

const table = document.createElement("table");

const thead = document.createElement("thead");

const tbody = document.createElement("tbody");

const headerRow = document.createElement("tr");

headers.forEach((h) => {

const th = document.createElement("th");

th.textContent = h;

headerRow.appendChild(th);

});

thead.appendChild(headerRow);

rows.forEach((row) => {

const tr = document.createElement("tr");

row.forEach((cell) => {

const td = document.createElement("td");

td.textContent = cell;

tr.appendChild(td);

});

tbody.appendChild(tr);

});

table.appendChild(thead);

table.appendChild(tbody);

return table;

}

/\*\*

\* Закрывает модальное окно аналитики.

\*/

function closeModal() {

document.getElementById("analyticsModal").style.display = "none";

}

/\*\*

\* Открывает Jupyter Notebook с предустановленным кодом для анализа CSV файла.

\*/

function openJupyterNotebook() {

const filename = document.getElementById("filename").value;

const code = `

import pandas as pd

import io

df = pd.read\_csv(io.BytesIO(uploaded['${filename}.csv']))

print(df)

`;

const jupyterLink = `https://jupyter.org/try?example=${encodeURIComponent(code)}`;

window.open(jupyterLink, '\_blank');

}

/\*\*

\* Открывает Google Colab с предустановленным кодом для анализа CSV файла.

\*/

function openGoogleColab() {

const filename = document.getElementById("filename").value;

const code = `

import pandas as pd

import io

df = pd.read\_csv(io.BytesIO(uploaded['${filename}.csv']))

print(df)

`;

const colabLink = `https://colab.research.google.com/notebook#create=true&code=${encodeURIComponent(code)}`;

window.open(colabLink, '\_blank');

}

/\*\*

\* Обрабатывает загрузку CSV файла.

\* @param {Event} event - Событие загрузки файла.

\*/

document.addEventListener("DOMContentLoaded", function () {

document

.getElementById("csvFileInput")

.removeEventListener("change", handleCSVFile);

document

.getElementById("csvFileInput")

.addEventListener("change", handleCSVFile);

});

/\*\*

\* Экспортирует данные в Excel.

\*/

async function exportToExcel() {

let csvContent = localStorage.getItem("csvData");

if (!csvContent) {

// Если данных нет в localStorage, загружаем их из IndexedDB

csvContent = await loadModelFromIndexedDB();

if (!csvContent) {

alert("Сначала создайте или загрузите CSV данные.");

return;

}

}

const processMessage = document.getElementById("processMessage");

const messageBox = processMessage.querySelector(".message-box");

// Устанавливаем текст сообщения

messageBox.textContent = "Сохраняем данные... Пожалуйста, подождите.";

// Отображаем контейнер

processMessage.style.display = "flex";

const csvLines = csvContent.split("\n");

const headers = csvLines[0].split(",");

const data = csvLines.slice(1).map((line) => line.split(","));

const workbook = XLSX.utils.book\_new();

const worksheet = XLSX.utils.aoa\_to\_sheet([headers, ...data]);

XLSX.utils.book\_append\_sheet(workbook, worksheet, "Sheet1");

const filename = document.getElementById("filename").value + ".xlsx";

XLSX.writeFile(workbook, filename);

processMessage.style.display = "none"; // Скрываем сообщение в случае ошибки

}

/\*\*

\* Экспортирует данные в JSON.

\*/

async function exportToJson() {

let csvContent = localStorage.getItem("csvData");

if (!csvContent) {

// Если данных нет в localStorage, загружаем их из IndexedDB

csvContent = await loadModelFromIndexedDB();

if (!csvContent) {

alert("Сначала создайте или загрузите CSV данные.");

return;

}

}

const processMessage = document.getElementById("processMessage");

const messageBox = processMessage.querySelector(".message-box");

// Устанавливаем текст сообщения

messageBox.textContent = "Сохраняем данные... Пожалуйста, подождите.";

// Отображаем контейнер

processMessage.style.display = "flex";

const csvLines = csvContent.split("\n");

const headers = csvLines[0].split(",");

const data = csvLines.slice(1).map((line) => line.split(","));

const jsonData = data.map((row) => {

const obj = {};

headers.forEach((header, index) => {

obj[header] = row[index];

});

return obj;

});

const jsonContent = JSON.stringify(jsonData, null, 2);

const filename = document.getElementById("filename").value + ".json";

const blob = new Blob([jsonContent], { type: "application/json" });

const url = URL.createObjectURL(blob);

const a = document.createElement("a");

a.href = url;

a.download = filename;

a.click();

URL.revokeObjectURL(url);

processMessage.style.display = "none"; // Скрываем сообщение в случае ошибки

console.log("Сохранено JSON содержимое:", jsonContent);

}

/\*\*

\* Обрабатывает событие изменения файла для сохранения CSV данных в IndexedDB и localStorage.

\* @param {Event} event - Событие изменения файла.

\*/

document.addEventListener("DOMContentLoaded", function () {

document.getElementById("csvFileInput")

.addEventListener("change", async (event) => {

const file = event.target.files[0];

if (file) {

const reader = new FileReader();

reader.onload = async (e) => {

const csvContent = e.target.result;

const maxLocalStorageSize = 5 \* 1024 \* 1024; // 5 MB

if (csvContent.length < maxLocalStorageSize) {

// Если данные меньше 5 MB, сохраняем в localStorage

localStorage.setItem("csvData", csvContent);

//alert('CSV данные сохранены в localStorage');

} else {

// Если данные больше 5 MB, сохраняем в IndexedDB частями

await saveModelToIndexedDBInParts(csvContent);

//alert('CSV данные сохранены в IndexedDB');

}

if (csvData && csvData.length > 0) {

generateCSV();

}

};

reader.readAsText(file);

}

});

});

/\*\*

\* Обрабатывает изменение типа индексов.

\*/

document.addEventListener("DOMContentLoaded", function () {

document.getElementById("indexType").addEventListener("change", function () {

const indexType = this.value;

const customIndexRange = document.getElementById("customIndexRange");

console.log("Выбранный тип индекса:", indexType);

if (indexType === "custom") {

customIndexRange.style.display = "block";

// customIndexRange.style.border = "2px solid red"; // Временно добавляем границу для отладки

console.log("Поле ввода пользовательских индексов отображается");

} else {

customIndexRange.style.display = "none";

console.log("Поле ввода пользовательских индексов скрыто");

}

});

});

/\*\*

\* Обрабатывает ввод пользователя и ищет совпадения в таблице.

\*/

async function searchRows() {

const userInput = document.getElementById("userInput").value;

const searchValues = userInput

.split(",")

.map((value) => value.trim().toLowerCase());

let csvContent = localStorage.getItem("csvData");

if (!csvContent) {

// Если данных нет в localStorage, загружаем их из IndexedDB

csvContent = await loadModelFromIndexedDB();

if (!csvContent) {

alert("Сначала создайте или загрузите CSV данные.");

return;

}

}

const csvLines = csvContent.split("\n");

const headers = csvLines[0].split(",");

const data = csvLines.slice(1).map((line) => line.split(","));

const matchingRows = data.filter((row) => {

return searchValues.some((value) =>

row.some((cell) => cell.toLowerCase().includes(value))

);

});

const searchResultsDiv = document.getElementById("searchResults");

searchResultsDiv.innerHTML = "";

if (matchingRows.length > 0) {

const totalRowsdata = data.length;

const matchPercentage = (matchingRows.length / totalRowsdata) \* 100;

const resultsTable = document.createElement("table");

const resultsThead = document.createElement("thead");

const resultsTbody = document.createElement("tbody");

const percentageDiv = document.createElement("div");

percentageDiv.textContent = `Найдено совпадений: ${

matchingRows.length

} (${matchPercentage.toFixed(2)}%)`;

percentageDiv.className = "percentage-info";

searchResultsDiv.appendChild(percentageDiv);

const headerRow = document.createElement("tr");

headers.forEach((header) => {

const th = document.createElement("th");

th.textContent = header;

headerRow.appendChild(th);

});

resultsThead.appendChild(headerRow);

matchingRows.forEach((row) => {

const tr = document.createElement("tr");

row.forEach((cell) => {

const td = document.createElement("td");

td.textContent = cell;

tr.appendChild(td);

});

resultsTbody.appendChild(tr);

});

resultsTable.appendChild(resultsThead);

resultsTable.appendChild(resultsTbody);

searchResultsDiv.appendChild(resultsTable);

} else {

const noResultsDiv = document.createElement("div");

noResultsDiv.textContent = "Совпадений не найдено.";

searchResultsDiv.appendChild(noResultsDiv);

}

}

**Функции генерации данных**

**addColumn:**

Добавляет новый столбец в форму.

Принимает параметры для названия столбца, типа данных и диапазона.

Параметры:

columnName (строка): Название столбца.

dataType (строка): Тип данных (например, "date", "time", "float\_range", "integer\_range", "range\_negative", "choice", "boolean", "name", "phone", "email", "random").

dataRange (строка): Диапазон данных (например, "2023.01.01-2023.12.31" для дат или "1-10" для чисел).

Возвращаемое значение:

Нет.

**removeColumn:**

Удаляет столбец из формы.

Принимает индекс столбца для удаления.

Параметры:

columnIndex (число): Индекс столбца для удаления.

Возвращаемое значение:

Нет.

**generateCSV:**

Генерирует CSV данные на основе введенных параметров.

Обрабатывает загруженные данные и добавляет новые столбцы.

Параметры:

Нет.

Возвращаемое значение:

Нет.

**generateData:**

Генерирует данные для столбца на основе типа данных и диапазона.

Поддерживает различные типы данных, такие как даты, время, числа, строки и т.д.

Параметры:

dataType (строка): Тип данных.

dataRange (строка): Диапазон данных.

numRows (число): Количество строк для генерации.

columnName (строка): Название столбца.

Возвращаемое значение:

Массив сгенерированных данных.

**Функции анализа данных**

**analyzeColumnData:**

Анализирует данные столбца на наличие уникальных и повторяющихся значений, а также пропущенных значений.

Возвращает метрики анализа данных.

Параметры:

columnName (строка): Название столбца для анализа.

Возвращаемое значение:

Объект с метриками анализа данных.

**showAnalytics:**

Отображает аналитику данных в модальном окне.

Анализирует данные и визуализирует результаты.

**Функции работы с данными**

**saveCSV:**

Сохраняет сгенерированные CSV данные в файл.

Использует localStorage для временного хранения данных.

**handleCSVFile:**

Обрабатывает загрузку CSV файла.

Парсит данные и отображает их в таблице.

**exportToExcel:**

Экспортирует данные в формат Excel.

Использует библиотеку XLSX для создания Excel файла.

**exportToJson:**

Экспортирует данные в формат JSON.

Преобразует CSV данные в JSON формат и сохраняет их в файл.

**Функции машинного обучения**

**createAndTrainModel:**

Создает и обучает модель машинного обучения на основе загруженных данных.

Использует библиотеку TensorFlow.js для создания и обучения модели.

**Функции взаимодействия с пользователем**

**checkAndPrompt:**

Проверяет и запрашивает ввод данных перед генерацией CSV.

Обеспечивает корректность введенных данных.

**closeModal:**

Закрывает модальное окно аналитики.

**openJupyterNotebook:**

Открывает Jupyter Notebook с предустановленным кодом для анализа CSV файла.

**openGoogleColab:**

Открывает Google Colab с предустановленным кодом для анализа CSV файла.

**clearTable:**

Полностью удаляет созданные данные и очищает хранилища.

**6.2. Документация и инструкции по использованию**

**Пользовательская документация**

**Введение**

CSV Generator — это веб-приложение, предназначенное для генерации, анализа и управления CSV данными. Оно позволяет пользователям создавать тестовые наборы данных с заданными параметрами и диапазонами, загружать и сохранять CSV файлы, анализировать данные и визуализировать результаты. Приложение также поддерживает интеграцию с машинным обучением, что позволяет создавать и обучать модели на основе загруженных данных.

**Основные функции и возможности приложения**

**Генерация данных:**

Создание данных различных типов (даты, время, числа, строки и т.д.) с заданными диапазонами и параметрами.

Возможность добавления и удаления столбцов, а также выбора типов данных и диапазонов.

**Загрузка и сохранение данных:**

Поддержка загрузки CSV файлов для анализа.

Возможность сохранения сгенерированных данных в форматах CSV, Excel и JSON.

**Анализ данных:**

Анализ данных на наличие пропусков, уникальных и повторяющихся значений.

Визуализация данных с помощью графиков и тепловых карт.

**Интеграция с машинным обучением:**

Возможность создания и обучения моделей машинного обучения на основе загруженных данных.

Прогнозирование и визуализация результатов модели.

**Удобный интерфейс пользователя:**

Интуитивно понятный интерфейс, который позволяет пользователям легко взаимодействовать с приложением и использовать его функции без необходимости глубоких технических знаний.

**Инструкции по установке и запуску приложения**

**Установка и запуск приложения**

**Скачайте проект:**

Скачайте архив с проектом и распакуйте его в удобное для вас место.

**Запустите локальный сервер:**

Откройте командную строку (или терминал) и перейдите в директорию проекта.

**Запустите локальный сервер с помощью команды:**

python -m http.server 8000

**Откройте браузер и перейдите по адресу:**

http://localhost:8000/index\_ls.html

**Использование приложения**

**Генерация данных:**

Введите количество строк и имя файла.

Выберите тип индексов.

Добавьте столбцы, задавая для каждого столбца название, тип данных и диапазон.

Нажмите кнопку "Создать CSV" для генерации данных.

**Загрузка данных:**

Нажмите кнопку "Загрузить CSV" и выберите файл для загрузки.

Данные отобразятся в таблице, и вы сможете выполнять различные операции с ними.

**Анализ данных:**

Нажмите кнопку "Показать аналитику".

В модальном окне вы увидите метрики анализа данных, такие как количество строк, столбцов, пропущенных значений и т.д.

Визуализация данных с помощью графиков и тепловых карт.

**Сохранение данных:**

Нажмите кнопку "Сохранить CSV", "Экспорт в Excel" или "Экспорт в Json" для сохранения данных в соответствующем формате.

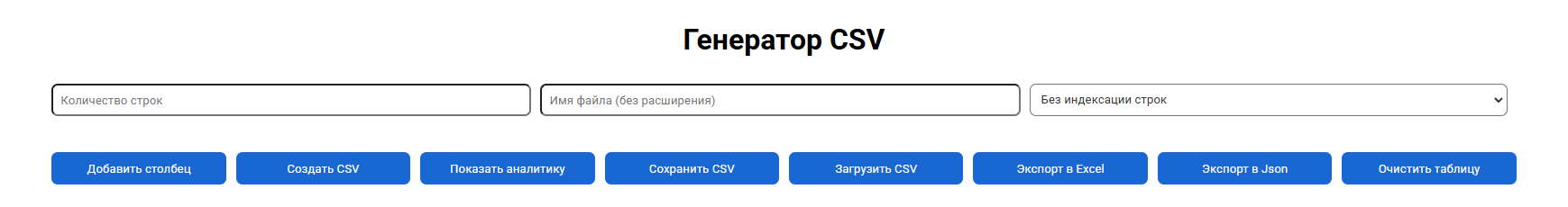
**Интеграция с машинным обучением:**

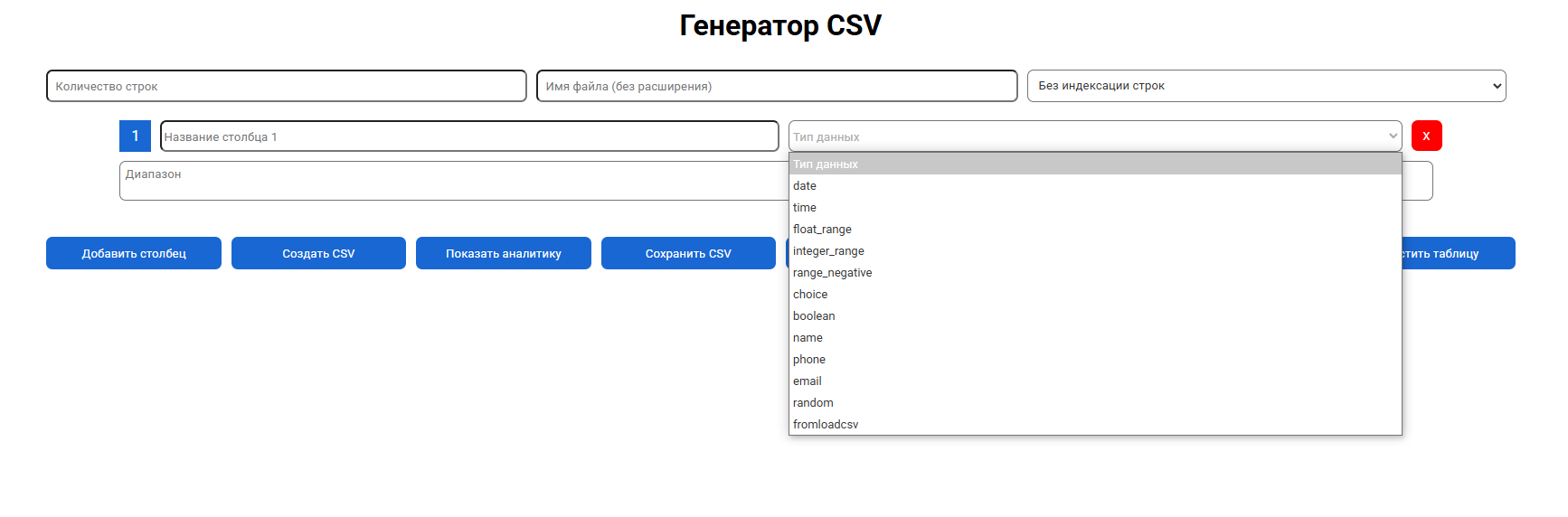
Используйте функцию создания и обучения моделей машинного обучения для анализа данных и прогнозирования.

**6.2. Примеры данных. Скриншоты и визуализации**

# Примеры данных, используемых в проекте.

# Примеры сгенерированных данных.





C:\Users\Veresova_AV\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Аннотация 2025-03-06 092641.png

