Санкт-Петербургский государственный университет

***ГОНЧАРОВ Даниил Юрьевич***

Выпускная квалификационная работа

# Реализация модуля «обследование датасета» в профилировщике данных Desbordante

Уровень образования: бакалавриат

Направление *02.03.03 «Технологии программирования»*

Основная образовательная программа *СВ.5162.2021 «Технологии программирования»*

Научный руководитель:
к. ф.-м. н., ст. научный сотрудник кафедры информационно-аналитических систем
Е. Г. Михайлова

Консультант:
ассистент кафедры информационно-аналитических систем Г. А. Чернышев

Рецензент:
инженер-разработчик, ООО «Открытая Мобильная Платформа», Бакалавр, Фирсов М. А.

Санкт-Петербург
2025

Saint Petersburg State University

***Daniil Hancharou***

Bachelor's Thesis

# Implementing dataset inspection subsystem in Desbordante data profiler

Education level: bachelor

Speciality *02.03.03 "Programming Technologies"*

Programme *CB.5162.2021 "Programming Technologies"*

Scientific supervisor:
C.Sc., Senior researcher E. G. Mikhailova

Consultant:
Assistant G. A. Chernishev

Reviewer:
senior developer at «Open Mobile Platform» M. A. Firsov

Saint Petersburg
2025

# Оглавление

# Введение

Каждый день человечество генерирует колоссальное количество информации, причём объём хранящихся данных по всему миру с каждым годом растёт экспоненциально и в настоящем 2025 году может превысить 175 зеттабайт [13]. Согласно исследованию [12], которое проводилось IBM, только для экономики США низкое качество данных приводит к потерям порядка трёх триллионов долларов в год. Данная информация заставила крупные организации переосмыслить классические подходы при работе с информацией и признать важность решения задач связанных с повышением качества данных. Это обуславливает растущий интерес к инструментам и методологиям в данной сфере.

Одним из ключевых этапов на пути к повышению качества данных является профилирование — сбор различной информации, относящейся к рассматриваемому набору данных. Текущий рынок предлагает огромное количество инструментов, которые позиционируются как приложения-профилировщики и предоставляют пользователям доступ к инструментарию для получения статистических значений, описывающих исследуемые данные. Польза полученных величин заключается в том, что они позволяют лучше понять структуру данных, выявить возможные корреляции и определить типы атрибутов. Однако, если пользователя интересует более глубокий анализ, позволяющий определять неявные зависимости, ему придётся использовать инструменты наукоёмкого профилирования, доступ к которым предоставляют далеко не все приложения.

Платформа Desbordante [4] позиционирует себя как наукоёмкий профилировщик данных. На её основе пользователям предоставляется инструментарий для проведения поиска и валидации различного рода зависимостей и схожих объектов: функциональных зависимостей [16], сопоставляющих зависимостей [14], ассоциативных правил [1] и многих других. Это позволяет глубже понимать природу исследуемых данных и делать более точные выводы. Однако, несмотря на доступ к реализованным алгоритмам по отдельности через различные интерфей-

сы (веб-приложение, интерфейс командной строки, Python-библиотека), Desbordante нельзя назвать профилировщиком в строгом смысле, так как просто передать набор данных на анализ и получить список конкретных фактов не предоставляется возможным. Для устранения этой проблемы было решено добавить в Desbordante модуль «обследования датасета». Также предполагается, что наличие подобного модуля в будущем поможет повысить востребованность Desbordante как продукта.

# 1. Постановка задачи

Целью данной работы является разработка и реализация приложения-профилировщика в рамках платформы Desbordante. Для её выполнения были поставлены следующие задачи:

1. Составить перечень требований к модулю профилирования данных.
2. На основе составленных требований разработать архитектуру.
3. Реализовать модуль профилирования данных.
4. Покрыть тестами кодовую базу.
5. Провести профилирование и апробацию на реальных данных.

# 2. Обзор

## 2.1. Глоссарий

- **Примитив**. Формализация понятия «шаблон» в данных. Как правило задаёт *класс* зависимостей (например, функциональных зависимостей, дифференциальных зависимостей) или других схожих объектов (например, ассоциативных правил). Выступает атомарным строительным блоком для использования в дальнейших программах.

  Различают *точные* и *приближённые* примитивы. Во втором случае в его определение вводится мера ошибки, допускающая частичные нарушения.

- **Экземпляр примитива**. Конкретный шаблон, принадлежащий классу, определяемому примитивом. Мы считаем, что экземпляр «удерживается» на наборе данных, если все записи набора удовлетворяют условиям примитива в пределах ошибки (если она задана).

- **Датасет**. Таблица с определённым количеством строк и столбцов в привычном понимании, представляет данные, к которым применяются алгоритмы поиска примитивов. В рамках данной работы датасетом считается сразу и исходный массив данных, и любой его поднабор.

## 2.2. Пример примитива

Данный раздел предназначен для демонстрации того, что мы относим к наукоёмкому профилированию и почему оно может быть интересно конечному пользователю. Будет рассмотрен один из примитивов [5], поиск которого реализован в рамках Desbordante.

*Определение*: Дифференциальная зависимость (DD) — это обобщение функциональной зависимости, в которой проверяется не равенство

Таблица 1: Пример набора данных о недвижимости (фрагмент)

| ID | Адрес | Тип дома | Спальни | Ванные |
|---|---|---|---|---|
| $t_1$ | Apt. 1603, 225 Handan Rd | Apartment | 1 | 1 |
| $t_2$ | Apt. 901, 225 Handan Rd | Apartment | 2 | 1 |
| $t_3$ | Apt. 901, 225 Handan Rd | Aparment[1] | 2 | 1 |
| $t_4$ | Unit 156, 899 Jiangwan Rd | Detached House | 5 | 3 |
| $t_5$ | Unit 222, 1555 Zhongqing Rd | Detached House | 8 | 5 |

значений, а разница между ними. Формально DD записывается как

$$\varphi_L[X] \ \rightarrow \ \varphi_R[A],$$

где $\varphi_L$ и $\varphi_R$ — дифференциальные функции. Говорят, что дифференциальная зависимость выполняется $R$, если для любых двух кортежей $r_1, r_2 \in R$ из того, что разность (дистанция) их значений по $X$ удовлетворяет ограничению $\varphi_L[X]$, следует, что разность их значений по $A$ обязана удовлетворять ограничению $\varphi_R[A]$.

Далее для наглядности рассматривается фрагмент набора данных (Таблица 1), содержащий информацию о недвижимости, а также дифференциальные зависимости, которые в нём наблюдаются.

- $\big[\mathit{Адрес}(\leq 0)\big] \rightarrow \big[\mathit{Тип\ дома}(\leq 1)\big]$: у домов по одному и тому же адресу тип отличается не более чем на один символ (устойчивость к опечаткам).
- $\big[\mathit{Тип\ дома}(\leq 1)\big] \wedge \big[\mathit{Спальни}(> 1)\big] \rightarrow \big[\mathit{Ванные}(> 2)\big]$: если различие по числу ванных больше одной, то различие по числу спален превышает две.

Дифференциальные зависимости устойчивы к шуму и опечаткам, а также помогают лучше определять дубликаты в записях. Это делает данный примитив полезным инструментом при профилировании и улучшении качества данных.

[1]Опечатка в данных создавшая дубликат.

## 2.3. Существующие решения

Перед началом разработки был рассмотрен текущий рынок инструментов, которые предоставляют доступ к профилированию данных. Это поможет выявить потенциальное место профилировщика Desbordante среди прочих решений.

Индустриальные продукты предлагают множество услуг по поддержанию и улучшению качества данных. Однако так как текущий вектор развития крупных компаний направлен в основном на технологии машинного обучения, задача профилирования данных в большинстве инструментов представлена не лучшим образом. В данной группе выделяются следующие:

- **IBM InfoSphere [7]** — мощный инструмент предоставляющий большое количество модулей, направленных на управление качеством данных. Благодаря возможности интеграции с другими решениями IBM позволяет решать внушительный объём задач различной направленности. Ориентирован в первую очередь на обработку больших данных и за счёт немалого инструментария позволяет проводить подробный анализ интересующих наборов данных.

- **Talend Open Studio [15]** — платформа предназначенная для комплексного управления большим объёмом данных. Предоставляет модули глубокой очистки и обработки информации. Также реализована поддержка проведения простого статистического анализа, который заключается в вычислении числовых характеристик для колонок.

- **Informatica Data Quality and Profiling [8]** — решение, которое в первую очередь предназначено на обнаружение потенциальных ошибок и аномалий в информации. Поддерживает большое количество инструментария по стандартизации данных внутри компаний, а также предлагает готовую библиотеку с обширным количеством бизнес-правил готовых к применению. Элемент профили-

рования — возможность проверять и находить заранее заданные шаблоны, выявлять функциональные зависимости, а также проводить статистический анализ.

Решения с открытым исходным кодом зачастую либо представляют собой инструменты общего назначения, либо фокусируются на решении конкретных задач по обеспечению качества данных. Наиболее популярные из них:

- **YData Profiling [18]** — библиотека Python, пользующаяся большой популярностью среди сообщества. Предлагает удобные способы обработки и анализ интересующих пользователя данных. Благодаря данному инструменту можно быстро получить статистическую информацию о датасете и продолжить анализ при помощи сторонних инструментов.

- **Data Cleaner [3]** — идея проекта заключается в облегчении процесса работы с данными для большего числа людей. Предлагает помощь в исправлении, насыщении и проведении статистического анализа данных.

- **Lux [10]** считается вспомогательным инструментом для представления полученной в процессе работы с ydata-profiling информацией в графическом виде. Имеет множество настроек, благодаря которым можно добиться различного рода визуализации данных. Также реализованы подсказки, которые помогают пользователям определиться с дальнейшими шагами анализа.

- **SweetViz [2]** — ещё одна Python-библиотека направленная на простоту анализа и визуализации информации. По заявлениям авторов предоставляет возможность провести профилирование и представить результат в виде чёткой и информативной визуализации благодаря двум строчкам кода. Под профилированием в данном случае подразумевается статистический анализ. Помимо этого реализован модуль сравнения разных наборов данных.

Отдельно стоит вынести решения с открытым исходных кодом, которые являются результатом исследовательских работ. Именно данный тип инструментов лучше всего подходит для сравнения с Desbordante, так как тоже направлен на наукоёмкое профилирование. Интересные представители:

- **Metanome [11]** — проект, которым вдохновлялся Desbordante на раннем этапе разработки. Предоставляет доступ к различным алгоритмам, реализующим поиск и валидацию широкого набора примитивов. Имеет несколько интерфейсов для удобства работы пользователей.

- **HoloClean [6]** — интересное решение, основной направленностью которого является улучшение и восстановление данных при помощи машинного обучения и минимального участия пользователя. В рамках данного обзора интересен тем, что в процессе работы определяет стандартные и условные функциональные зависимости.

- **Uni-Detect [17]** — находит потенциальные ошибки и неточности в данных при помощи определения функциональных зависимостей и «what-if» анализа.

## 2.4. Выводы

Обзор показывает, что рынок приложений-профилировщиков насыщен различными решениями. Пользователи могут с минимальными усилиями провести анализ своих данных, сравнить с другими или перевести в красивый визуальный вид, провести очистку. Компании в свою очередь могут получить практически полное покрытие задач, возникающих при работе с данными.

Однако в процессе прочтения описаний рассмотренных инструментов можно заметить, что анализ, который предлагается большинством, зачастую статистический, и выявить в данных зависимости сложнее

функциональных становится непростой задачей. Проблема же исследовательских прототипов состоит в прекращении поддержки (за малым исключением), а также трудной доступности к реализованному функционалу. Это значит, что профилировщик в рамках платформы Desbordante сможет найти свою нишу даже среди такого богатого рынка решений.

# 3. Требования

Следует понимать, что поиск примитивов является трудоёмкой задачей, поэтому вследствие нехватки памяти при выполнении нередко могут возникать сбои.

С учётом данного аспекта для модуля профилирования данных в рамках платформы Desbordante в ходе обсуждений был сформулирован следующий список требований:

1. Сценарий профилирования должен храниться в виде «профиля», который должен отражать набор примитивов, поиск которых будет произведён, а также быть настраиваемым

2. Должен существовать механизм реакции на сбои, который будет корректно обрабатывать ошибки в процессе выполнения алгоритмов.

3. Необходимо обеспечить хранение информации о проведённых запусках. Информация должна содержать точную причину сбоя, если поиск примитива завершился неудачей.

4. Необходим механизм, позволяющий не производить повторные вычисления задач, для которых уже был найден результат.

5. Алгоритмы должны корректно работать с заданным временным ограничением и завершаться при его истечении.

6. При профилировании должна вестись подробная запись процесса выполнения.

7. Найденная в процессе работы информация должна храниться в человекочитаемом и машиночитаемом виде.

Именно на основе этих требований проводилась дальнейшая реализация модуля профилирования данных.

# 4. Реализация

Было решено использовать подход, который уже применялся в Desbordante при разработке интерфейса командной строки. Идея заключается в предоставлении конечному пользователю доступа к алгоритмам, реализованным на C++, при помощи промежуточного слоя — python-привязок (Рисунок 1).

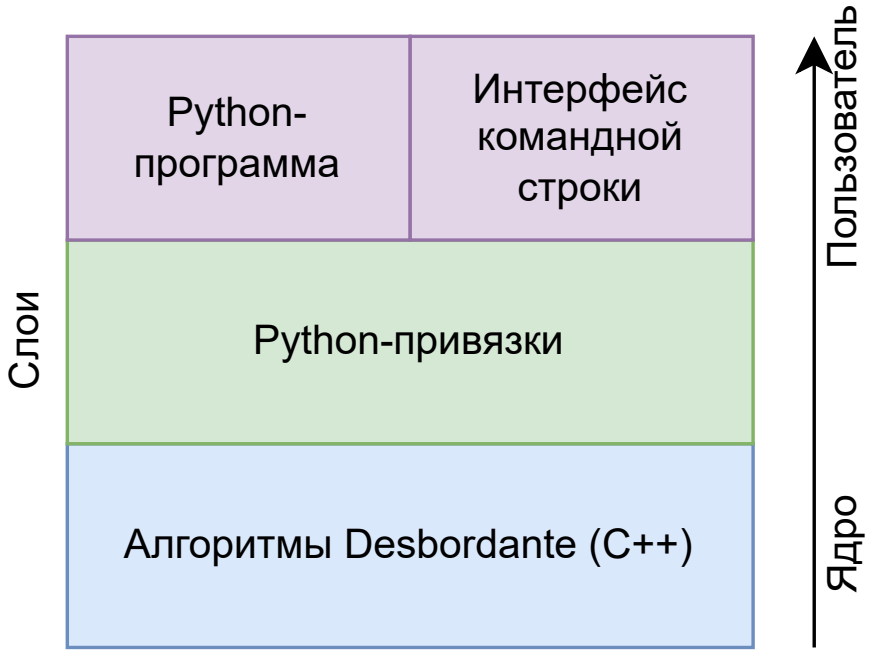


Рис. 1: Взаимодействие пользователя с ядром Desbordante.

В данном разделе будет представлено описание решений, которые были приняты для создания профилировщика на основе Python-модуля desbordante.

## 4.1. Архитектура

Для основной логики будущего профилировщика была предложена модульная архитектура, где каждый модуль имеет конкретное назначение. С учётом составленного списка требований и представления о примерном ходе работы приложения удалось выделить 9 компонентов:

- **Модуль загрузки данных**. Отвечает за загрузку и представления датасета в удобном виде. Необходим для вынесения логики предобработки исследуемого набора данных.

- **Модуль загрузки профилей**. Переводит заранее заданный «сценарий» профилирования из человекочитаемого вида в список конкретных задач.

- **Модуль алгоритмов**. Модуль доступа ко всем необходимым алгоритмам Desbordante через унифицированный интерфейс. Нужен для постобработки и возможной предобработки данных, которые поступают в алгоритмы и получаются на выходе.

- **Модуль координации**. Обработчик списка задач, полученного из модуля загрузки профилей. Посылает их на выполнение и в зависимости от результата может сформировать новый список задач, являющийся подмножеством предыдущего.

- **Модуль диспетчеризации задач**. Получает на вход список задач и запускает их, контролируя выполнение. Именно тут отслеживаются потенциальные лимиты, а также сбои при работе алгоритмов.

- **Модуль реакции на сбой**. Модуль, который ответственен за принятие решений в случае сбоев при работе алгоритмов. Помогает в формировании нового списка задач.

- **Модуль истории запусков**. Модуль записи и чтения информации о запусках в рамках приложения.

- **Модуль сравнения**. Отвечает за логику сравнения результатов запуска для двух различных наборов данных. Предполагаемые сценарии сравнения: один датасет является частью другого, один датасет является новой версией другого.

- **Модуль запуска**. Является точкой входа в систему и содержит API для работы с логикой приложения-профилировщика. Нужен

для связи остальных модулей в единый процесс.

Диаграмма компонентов демонстрирующая примерное взаимодействие между описанными модулями представлена в работе (Рис. 2).

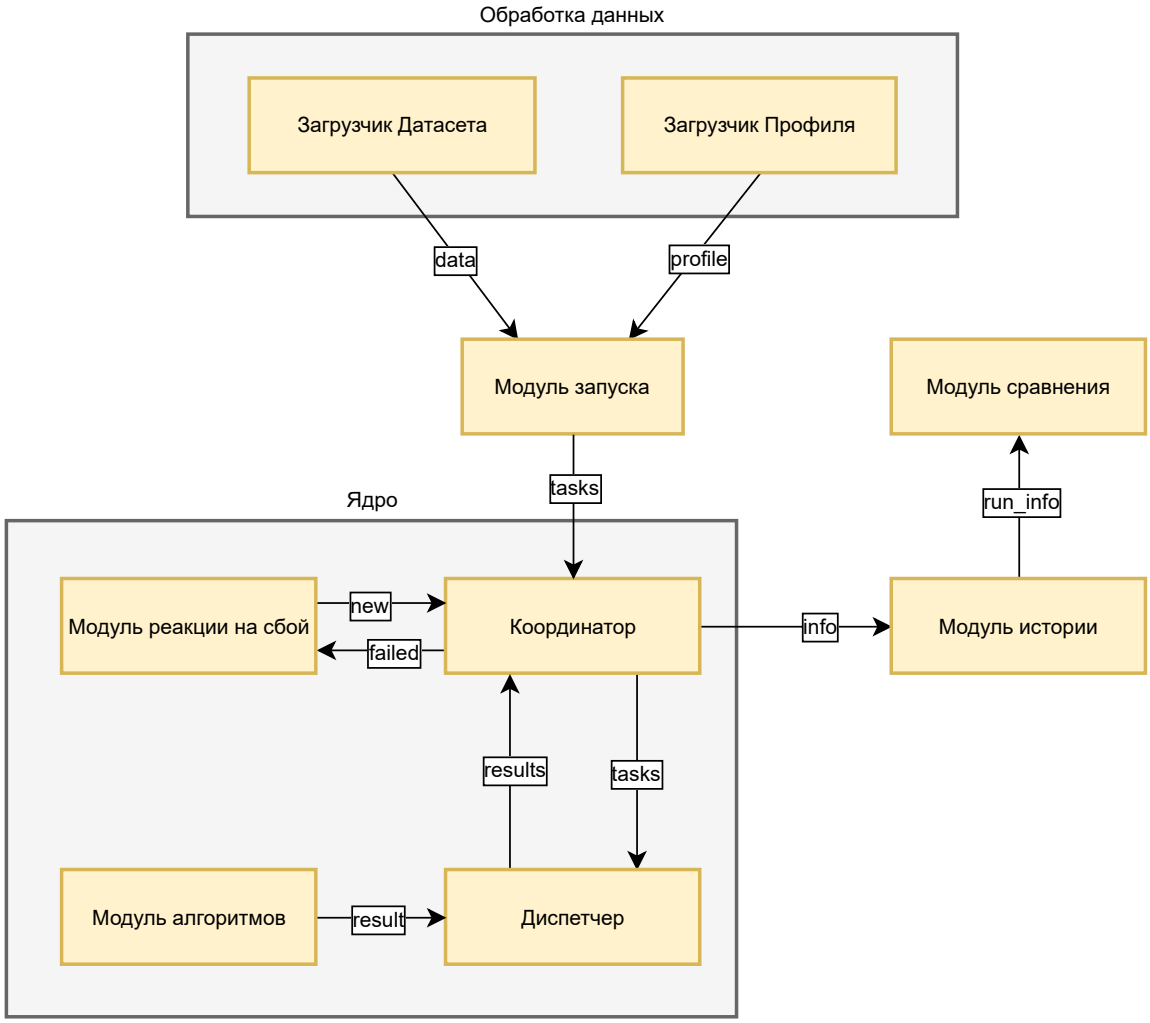


Рис. 2: Диаграмма компонентов.

## 4.2. Обработка данных

**Модуль загрузки данных**. Содержит две функции: calculate_data_hash и get_dataframe_and_hash. Первая вычисляет значение хеш-функции для переданного файла, вторая в свою очередь представляет переданный файл в виде объекта pandas.DataFrame,

а также возвращает вычисленное первой функцией хеш-значение. Рассмотрим их отдельно.

calculate_data_hash принимает в качестве параметров путь к файлу, имя хеш-алгоритма[2], а также размер блока[3] для загрузки. Суть работы очень простая: при помощи модуля hashlib и переданного имени создаётся соответствующий хеш-алгоритм. Далее идёт чтение переданного файла блоками указанного размера, данный механизм позволяет обрабатывать даже объёмные файлы, которые не всегда способны уместиться в памяти сразу. После обработки каждого блока хеш-значение обновляется и в завершении получается необходимый результат.

Само хеш-значение необходимо для идентификации запусков на одном и том же датасете. Это делает возможным реализацию механизма «кэширования», который позволяет не запускать алгоритм на наборе данных, а просто десериализовать хранящийся результат, если таковой имеется.

В свою очередь get_dataframe_and_hash имеет следующие параметры: путь к файлу, разделитель, наличие заголовка, количество столбцов и количество колонок. При помощи первых трёх, а также функции pandas.read_csv создаётся объект pandas.DataFrame, который затем «обрезается», если переданное значение столбцов или колонок меньше текущего. Далее при помощи описанной выше функции вычисляется хеш-значение и на выход подаётся соответствующий кортеж.

**Модуль загрузки профилей**. Стояла задача выбрать формат, в котором будут описываться потенциальные сценарии профилирования. Необходимо было найти вариант удобный для хранения и редактирования. Набор, среди которого проводился выбор: INI, YAML, JSON, XML.

При описании задач предпочтительно было бы иметь гибкость, поэтому от формата INI, который не поддерживает вложенные структуры, было решено отказаться.

Из-за трудной читаемости также решено было отказаться и от XML.

[2]Стандартно выбран алгоритм SHA256, так как автор работы посчитал, что SHA512 будет излишним для данной задачи.

[3]Размер блока влияет только на процесс получения, а не на само хеш-значение. В качестве стандартного было выбрано значение, которое является оптимальным на взгляд автора: 64 KB.

Для оставшихся кандидатов представленных YAML и JSON были написаны примеры описания одинакового сценария профилирования: Листинг 1 и Листинг 2 соответственно. На взгляд автора вариант с YAML оказался более простым и понятным в использовании, поэтому данный формат решено было использовать для задания профилей.

**Листинг 1: Пример задания профиля при помощи YAML.**

```yaml
name: "short_analysis"
global_settings:
  rows: 50000
  columns: 10
tasks:
  - family: "afd"
    parameters:
      error: 0.1
    timeout: 60

  - family: "ucc"
    timeout: 60
```

В самом коде профиль представляется объектом класса Profile, который в качестве полей содержит имя, список задач для запуска и словарь global_settings, в котором может задаваться количество строк и столбцов для анализа, а также общий лимит времени.

Задачи из списка описываются классом TaskProfile. Поля данного класса описывают характерные для каждой задачи величины: примитив, алгоритм, параметры и временной лимит. Как было видно в примере задания профиля (Листинг 1), для задачи можно не указывать алгоритм, если указан примитив. В таком случае в качестве алгоритма будет выбран стандартный представитель для указанного примитива. Это реализовано через функцию get_algorithm_name_by_family. Аналогичным образом можно не указывать примитив, если указан алгоритм — функция get_family_by_algorithm; выбор между обычными и приближёнными вариантами примитивов в данном случае осуществляется на основе значения параметра error.

Сам процесс распознавания содержимого YAML файла происходит с помощью модуля yaml в функции load_profile, которая принимает путь к файлу в качестве параметра.

**Листинг 2: Пример задания профиля при помощи JSON.**

```
{
"name": "short_analysis",
"global_settings": {
    "rows": 50000,
    "columns": 10
}
"tasks": [
    {
    "family": "afd",
    "parameters": {
        "error": 0.1
    },
    "timeout": 60
    }
]
}
```

## 4.3. Запуск алгоритмов

**Модуль алгоритмов**. По сути представлен двумя подмодулями, так как desbordante предоставляет доступ и к алгоритмам поиска, и к алгоритмам валидации. Поскольку в приложении-профилировщике используются алгоритмы обоих видов, необходимо было сделать унифицированные обёртки для каждого из них.

Рассмотрим алгоритмы поиска. Реализован интерфейс MiningAlgorithmInterface, который содержит четыре абстрактных метода: load_data, execute, get_results и run. Метод run объединяет load_data, execute и get_results для удобного использования, поэтому в дальнейшем возвращаться к нему не будем.

Для каждого примитива реализован свой класс, который наследуется от описанного интерфейса и переопределяет его методы. Поля parameters и instance определяют параметры запуска и экземпляр алгоритма соответственно. Для примитивов с несколькими алгоритмами добавлено поле algorithm_name, которое, благодаря соответствующим словарям (FD_ALGO_MAP, UCC_ALGO_MAP и так далее), создаёт в конструкторе класса нужный экземпляр.

load_data переопределяется для загрузки датасета в алгоритм. В большинстве случаев делается это при помощи

algo.load_data(table=dataframe), однако есть алгоритмы, для которых загрузка немного отличается, например для HyMD: algo.load_data(right_table=dataframe). Датасет передаётся как параметр.

Метод execute запускает алгоритм на загруженном датасете с выбранными параметрами и для всех алгоритмов имеет одинаковый вид: algo.execute(**self.parameters).

Метод get_results предназначен для составления словаря результата. Ключами данного словаря являются названия примитивов или других сущностей, полученных в результате работы алгоритма, а списки найденных экземпляров хранятся в качестве значений.

Алгоритмы валидации добавлены схожим образом. Реализован интерфейс VerificationAlgorithmInterface, который содержит три абстрактных метода: load_data, get_broken_instances и run, где load_data и run аналогичны алгоритмам поиска.

Аналогичным образом для каждого примитива реализован собственный класс. В отличие от случая с алгоритмами поиска, количество полей у данных классов всегда равно единице и соответствует экземпляру алгоритма.

Метод get_broken_instances принимает в качестве параметра список экземпляров соответствующего примитива и для каждого элемента запускает валидацию, формируя в процессе список «сломанных» экземпляров. Данный список и подаётся на выдачу.

Стоит отметить, что на момент написания работы алгоритмов валидации в Desbordante значительно меньше[4] по сравнению с алгоритмами поиска, однако это развивающийся вектор в рамках платформы, так что со временем размеры данных семейств должны сравняться.

**Модуль диспетчеризации задач**. Центральный элемент системы, который отвечает за эффективное и контролируемое выполнение списка задач профилирования.

В данном модуле определён класс TaskToRun, который описывает

[4]Речь идёт об алгоритмах валидации примитивов, для которых реализованы алгоритмы поиска. В данной работе не рассматривались отдельные примитивы, для которых реализован только алгоритм валидации.

задачу для запуска и хранит информацию, которая может пригодиться на различных этапах обработки. Поля данного класса содержат подробную информацию о задаче: уникальный идентификатор задачи для истории запусков, имя примитива и алгоритма, параметры, информацию о датасете, ограничение по времени, стратегию обработки исключения (подробнее в разделе реакции на сбой), а также текущую итерацию данной задачи. Практически все модули оперируют именно с объектами класса TaskToRun.

Вернёмся к работе самого модуля, она основана на следующих принципах:

- **Управление ресурсами и мониторинг**. Пользователь может задать ограничение по памяти, количеству доступных ядер, ограничение по времени на конкретную задачу, а также глобальное ограничение по времени на весь список задач текущей итерации.

  Диспетчер в свою очередь устанавливает данные ограничения на запускаемые процессы и отслеживает их выполнение.

- **Параллельное вычисление**. По умолчанию алгоритмы «пытаются» запускаться в параллель. Это значит, что на первой итерации доступные ресурсы равномерно распределяются между процессами, данный подход позволяет сократить время профилирования на многоядерных системах. Однако стоит помнить, что представленные в desbordante алгоритмы реализуют трудоёмкие задачи, так что часто некоторым процессам не хватает памяти и тогда они отправляются на «второй круг».

  Все итерации начиная со второй работают не в параллель, в данном режиме всеми доступными ресурсами владеет единственный процесс. Это позволяет успешно выполнять алгоритмы с высоким потреблением памяти.

  У пользователя есть возможность отключить попытку параллельного вычисления и сразу начинать с одиночного.

- **Изоляция и отказоустойчивость**. Каждая задача выполняется

в отдельном процессе, что обеспечивает её изоляцию, поэтому возможные ошибки при выполнении не влияют на выполнение других. Для экстренных случаев реализован метод terminate_process, который обеспечивает корректное завершение процессов и дочерних элементов.

В работе схематично представлено устройство диспетчера задач (Рис. 3).

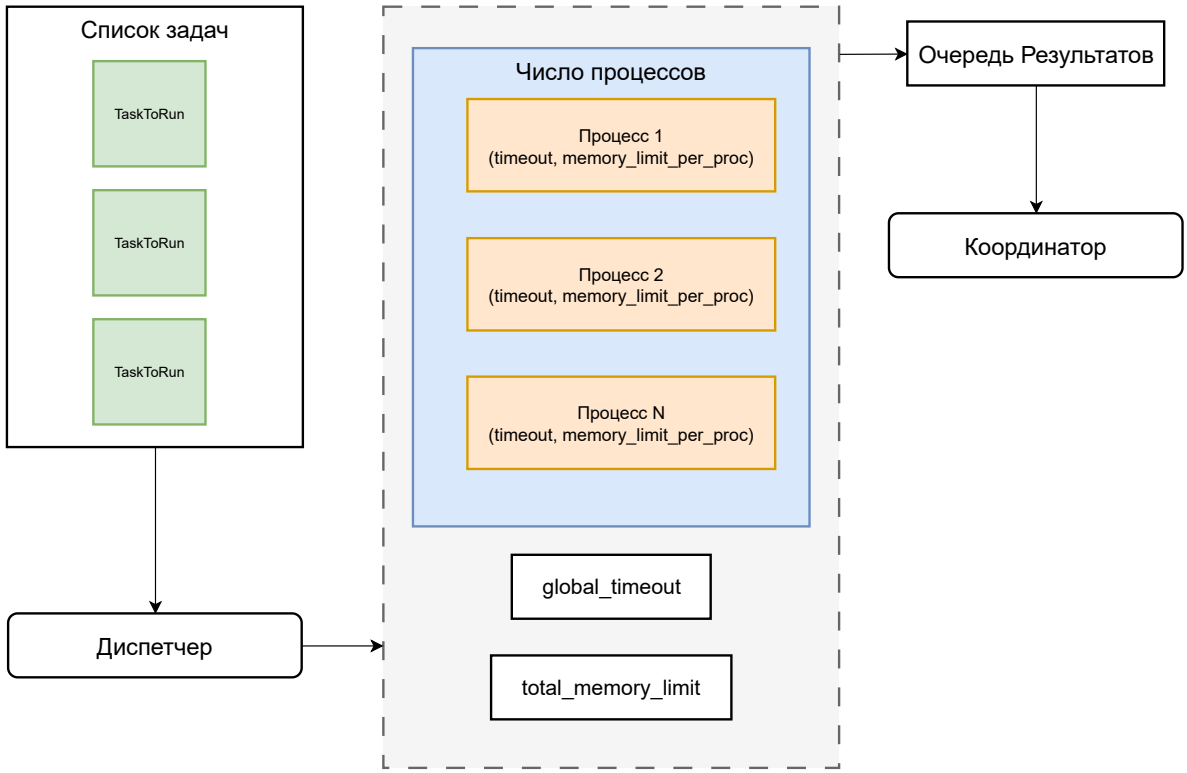


Рис. 3: Представление работы диспетчера.

**Модуль координации**. Основной метод данного модуля — execute_tasks_to_run. Как и следует из названия, данный метод вызывает модуль диспетчеризации и передаёт ему подготовленный список задач для выполнения. Перед этим вызывается метод check_existing_results, который при помощи описанного ранее функционала модуля истории запусков проверяет наличие сохранённого результата для каждой из задач. Если такой результат имеется, то его значение десериализуется, а задача удаляется из списка.

После выполнения работы модуля диспетчеризации, структура которого была описана ранее, происходит обработка результатов — метод process_results. Внутри данного метода происходит распаковка значений, которые вернулись из запущенных процессов, и на их основе определяются дальнейшие действия. Если результат успешный, то через метод store_result найденные экземпляры выводятся в файл и сериализуются для потенциального будущего использования. Если же процесс закончился сбоем, то вызывается метод handle_task_failure. Тот в свою очередь на основе решения модуля реакции на сбой определяет, что будет происходить с провалившейся задачей:

- Пропуск;
- Повтор;
- Повтор с изменениями: новое временное ограничение или другой размер датасета.

Если решено повторить запуск задачи (с изменениями или без), то при помощи метода create_new_task создаётся новый объект TaskToRun и помещается в новый список задач для запуска.

Помимо описанных, также реализованы методы, необходимые для записи и обновления информации в истории запусков.

**Модуль реакции на сбой**. Представляет из себя одну функцию, которая предназначена для обработок неудачных запусков алгоритмов. Решение принимается на основе типа ошибки, а также выбранной из списка стратегии:

- **Рост временного ограничения**. Данная стратегия предназначена для выявления времени, за которое алгоритмы обработают заданный датасет. Следовательно, если тип ошибки Timeout, то задача запускается заново с увеличенным временным ограничением. Используемые параметры: timeout_step и max_limit. Предназначены соответственно для определения величины, на которую увеличивается ограничение за одну стадию, а также максимально

возможный лимит, при достижении которого новых итераций не происходит.

- **Сокращающий поиск**. Эта стратегия в свою очередь выявляет количество строк, которое корректно обрабатывается алгоритмами. Если тип ошибки Timeout или MemoryError, то область поиска сокращается и задача запускается заново. Параметры: prune_factor и min_rows. Первый определяет коэффициент, на который умножается текущее количество строк для определения новой области поиска, а второй указывает минимальное количество строк, при достижении которого сокращение заканчивается и задача перестаёт перезапускаться.

- **Спросить пользователя**. Как и следует из названия, при данной стратегии пользователь сам решает, что делать с провалившимися задачами: пропустить, перезапустить или сократить датасет и перезапустить.

- **Автоматический**. При данном режиме профилировщик самостоятельно пытается добиться выполнения задачи. Первым делом будет предложен перезапуск, если ошибка не исчезнет, он постарается сократить область поиска, а если и это не поможет, то задача будет пропущена.

На выход подаётся следующий набор: действие (пропустить, повторить) и новые параметры для повтора, если они нужны.

**Модуль сравнения**. Представляет дополнительную задачу, поддерживаемую профилировщиком desbordante: сравнение результатов профилирования двух датасетов. На данный момент представлен сценарий, где датасет является частью большего, а также где один датасет является новой версией другого.

Суть состоит в том, что сначала запускается стандартное профилирование для отдельных наборов данных, а затем для каждого примитива из сценария происходит сравнение: определяются новые экземпляры и экземпляры, которые соответствуют второму датасету. Функция

сравнения — get_runs_comparison_analyze.

Именно в данном модуле происходит использование алгоритмов валидации, так как если какая-то задача провалилась на большем датасете, но успешно выполнилась на меньшем и для данного примитива существует алгоритм валидации, сравнение происходит именно на основе валидации.

## 4.4. Информация о запусках

**Модуль истории запусков**. Представляет собой класс HistoryStorage, содержащий единственное поле filename, хранящее имя JSON файла, в котором ведётся запись истории запусков. Помимо методов инициализации, загрузки и сохранения реализует метод добавления записи — add_run, метод обновления записи — update_run, метод записи успешного выполнения — mark_success и метод записи ошибки при выполнении — mark_failure. В качестве параметров данные методы принимают словарь task_info, содержащий информацию, которую необходимо отобразить в истории запусков.

Кроме этого реализованы методы get_tasks_by_run_id и get_last_run_for_algo_and_data. Первый принимает run_id и возвращает информацию о задачах, относящихся к соответствующему запуску в виде списка. Второй в свою очередь в качестве параметров принимает имя алгоритма, список параметров и хеш-значение датасета и возвращает информацию о последнем успешном подходящем запуске, если такой хранится в истории.

По умолчанию запись истории ведётся в файле history.json. В работе представлен примерный[5] фрагмент файла истории запуска (Листинг 3).

**Логирование**. Также был реализован небольшой модуль для логирования, который не описывался в архитектуре. Реализация строится на стандартном для языка Python модуле logging. Добавлено три функции:

[5]Значения некоторых полей сокращены для простоты восприятия, на понимание структуры записи это не влияет.

**Листинг 3: Фрагмент файла истории запусков.**

```
{
"runs": [
    {
    "run_id": "b725c3ab-06a2-484b-b253-fe8d9a55bfbf",
    "task_id": "4df7d195-8302-46ae-97c6-22537943bf45",
    "algorithm": "pyro",
    "algorithm_family": "afd",
    "params": {
        "error": 0.1,
        "threads": 4
    },
    "data_hash": "hash-value",
    "rows": 10000,
    "cols": 7,
    "timestamp_start": 420.30,
    "result": "Success",
    "timestamp_end": 420.49,
    "execution_time": 0.19,
    "result_path": "pyro.pkl",
    "instances": 17
    }
]
}
```

1. **configure_core_logger**: устанавливает переданный уровень логирования. По умолчанию это DEBUG.

2. **add_console_handler**: добавляет новый обработчик в виде потока вывода данных. Стандартное значение для уровня логирования: INFO.

3. **add_file_handler**: добавляет новый обработчик в виде переданного файла. Стандартное значение для уровня логирования: DEBUG.

Такое разделение по уровням логирования обусловлено идеей, что при стандартной работе для пользователя выводится только необходимая и понятная информация, а все технические подробности хранятся в файле логирования соответствующего запуска. При желании пользователь может изменить уровень логирования, отображаемый в консоли.

## 4.5. Запуск приложения

**Модуль запуска**. Как и было сказано в разделе с архитектурой, данный модуль предназначен для внешнего взаимодействия с логикой профилировщика. Содержит функции для запуска профилирования датасета, запуска сравнения с меньшим датасетом, а также запуска сравнения с новой версией.

Помимо этого содержит вспомогательные функции для генерации отчётов и создания директорий отдельных запусков

**Пользовательский интерфейс**. В качестве интерфейса взаимодействия был выбран интерфейс командной строки. Он был реализован с помощью Python-библиотеки Click. Профилирование запускается командой run, а сравнение командной compare subset или compare version.

Также добавлено текстовое описание режимов и всех доступных опций. Это предоставляет пользователю возможность получить вспомогательную информацию при помощи флага –help.

## 4.6. Сериализация

Отдельно выделена реализация механизма сериализации и десериализации объектов, являющимися результатом работы запускаемых алгоритмов.

В рамках платформы Desbordante алгоритмы реализуются на языке C++, а Python-привязки и собственно сам Python-модуль desbordante генерируется при помощи библиотеки pybind11. Из-за этого возвращаемые алгоритмами объекты (desbordante.FD, desbordante.DD, desbordante.UCC и так далее) содержат в своей структуре нетипичные для языка Python типы данных, что в свою очередь делает невозможным поддержку сериализации стандартным модулем pickle.

Решение данной проблемы описано в документации pybind11 [9]. Для каждого объекта, который является результатом работы какого-либо алгоритма необходимо определить методы getstate и setstate. Первый метод переводит поля передаваемого C++ объекта в некий кортеж значений, содержащий данные в интерпретируемом для языка Python

виде, а второй на основе этого кортежа создаёт C++ объект, который в идеале должен быть равен изначальному.

Реализация методов getstate и setstate была добавлена[6] в Python-привязки для всех используемых алгоритмов.

[6]Изменения кода основного репозитория отражены в pull request с номером 542

# 5. Процесс работы с приложением

## 5.1. Сценарии использования

В начале данного раздела кратко рассмотрим примерный сценарий профилирования с использованием существующих интерфейсов. Так как в реализуемом инструменте взаимодействие с пользователем реализовано через интерфейс командной строки, для примера возьмём соответствующий интерфейс Desbordante.

Для нашего небольшого примера в сценарий профилирования включим три задачи:

1. Поиск AFD;
2. Поиск AUCC;
3. Поиск DD.

Запуск соответствующих алгоритмов происходит посредством следующих вызовов:

```
python3 cli.py --table=data.csv --task=afd --error=0.1

python3 cli.py --table=data.csv --task=aucc --error=0.1

python3 cli.py --table=data.csv --task=dd
```

Если мы захотим применить такой же сценарий к другому датасету, нам придётся изменить значение table и сделать такое же количество повторных вызовов. То есть при увеличении числа задач, будет увеличиваться и число необходимых вызовов для каждого нового процесса профилирования.

Реализованный модуль позволяет решить данную проблему. Рассмотрим аналогичный сценарий с тремя задачами. Составим новый профиль, который описывает интересующие нас задачи (Листинг 4).

**Листинг 4: Профиль с AFD, AUCC и DD.**

```yaml
name: "example_profile"
tasks:
  - family: "afd"
    parameters:
      error: 0.1

  - family: "aucc"
    parameters:
      error: 0.1

  - family: "dd"
```

После этого дальнейший процесс профилирования состоит из вызова команды run профилировщика:

```
python3 desbordante_profiler.py run --data=data.csv
↪  --profile=example_profile.yaml
```

Если мы захотим провести профилирование нового датасета, необходимо будет поменять только значение data:

```
python3 desbordante_profiler.py run --data=new_data.csv
↪  --profile=example_profile.yaml
```

Теперь допустим, что new_data.csv является обновлённой версией data.csv и нам необходимо узнать потенциальные изменения. Для этого реализована команда compare version. Примерный вызов выглядит следующим образом:

```
python3 desbordante_profiler.py compare version
↪  --initial=data.csv --target=new_data.csv
↪  --profile=example_profile.yaml
```

Команда compare subset реализует схожий функционал, но для случая, когда один датасет является меньшей версией другого:

```
python3 desbordante_profiler.py compare subset
↪  --subset=data_half.csv --target=data.csv
↪  --profile=example_profile.yaml
```

**Листинг 5: Каталог results.**

```
results/
├── comparison_data1k_data5k_profile_YYYY-MM-DD-HH-MM-SS/
│   ├── profiling_data1k_profile_YYYY-MM-DD-HH-MM-SS/
│   ├── profiling_data5k_profile_YYYY-MM-DD-HH-MM-SS/
│   ├── comparison.log
│   ├── results.txt
│   └── digest.md
└── profiling_data_profile_YYYY-MM-DD-HH-MM-SS/
    ├── serialized_data/
    ├── profiling.log
    ├── results.txt
    └── digest.md
```

## 5.2. Каталог результатов

Результаты хранятся в отдельном каталоге results. Для каждого запуска профилирования создаётся собственная папка, которая содержит папку с результатами в машиночитаемом виде, файл с результатами в текстовом виде, файл с логами и файл отчёта.

Для compare-запусков создаётся папка, содержащая папки запусков профилирования для каждого датасета и аналогичные файлы с результатами, логами и отчётом.

Примерный вид каталога results представлен в работе (Листинг 5).

# 6. Тестирование и апробация

Было реализовано шесть наборов тестов, пять из которых тестируют компоненты приложения в изоляции, а шестой тестирует запуск от начала до конца. При помощи инструментов GitHub запуск описанных тестов был добавлен в механизм непрерывной интеграции. Это позволит отлавливать потенциальные ошибки при будущей работе над проектом.

Для проверки общей работоспособности был составлен сценарий профилирования, включающий следующие задачи:

1. Поиск MD;
2. Поиск NAR;
3. Поиск FD;
4. Поиск AFD;
5. Поиск UCC;
6. Поиск AUCC.

Значение error для приближённых алгоритмов было задано как 0.05, остальные параметры ставились по умолчанию.

Данный сценарий профилирования был применён к наборам данных, информация о которых содержится в соответствующей таблице (Таблица 2). digits.csv использовался дважды с разным значением количества строк.

Таблица 2: Датасеты для профилирования.

| Датасет | Столбцов | Строк |
|---|---|---|
| adult.csv | 15 | 32560 |
| digits_mini.csv | 10 | 1797 |
| digits.csv | 25 | 1797 |
| neighbors10k.csv | 7 | 9999 |

Таблица 3: Результаты профилирования набора датасетов.

| Датасет | MD | NAR | FD | AFD | UCC | AUCC | Время(с) |
|---|---|---|---|---|---|---|---|
| adult.csv | 2 | 66 | 201 | 2 | 0 | 219 | 19.83 |
| digits_mini.csv | 1 | 142 | 54 | 1 | 0 | 0 | 0.51 |
| digits.csv | 9 | 0 | 1653 | 9 | 641 | 146 | 9.54 |
| neighbors10k.csv | 16 | 187 | 16 | 12 | 3 | 3 | 8.85 |

Таблица 4: Результат полного профилирования.

| Датасет | CFD | FD | AFD | IND | UCC | AUCC | OD |
|---|---|---|---|---|---|---|---|
| neighbors100.csv | 50 | 9 | 9 | 2 | 2 | 2 | 9 |
| neighbors100.csv | DD | MD | NAR | DC | AR | AC | - |
| neighbors100.csv | 1 | 4 | 159 | 8 | 50 | 33 | - |

В результате проведённого профилирования был получен результат, который представлен в таблице (Таблица 3). Для примитивов указано количество найденных экземпляров. Также указано общее время выполнения. Стоит отметить, что для датасетов с большим количеством строк (adult.csv и neighbors10k.csv) основное время работы отведено алгоритмам поиска MD и DD. Для digits.csv основное время было потрачено на вычисление FD, AFD, UCC и AUCC.

Также был составлен сценарий профилирования, содержащий все поддерживаемые примитивы. В качестве набора данных использовался срез датасета neighbors.csv в 100 строк. Результат представлен в соответствующей таблице (Таблица 4).

В качестве апробации были рассмотрены экземпляры примитивов для датасета adult.csv, которые были найдены в процессе профилирования. Это позволило выявить следующие интересные факты:

1. Функциональные зависимости

   $$\textit{education} \rightarrow \textit{education-level} \quad \text{и} \quad \textit{education-level} \rightarrow \textit{education}$$

   показывают, что числовое представление уровня образования полностью дублирует текстовое. Это говорит о том, что наличие раз-

ных видов представления излишне, поэтому одно из них можно убрать без потери информации.

2. Приближённая функциональная зависимость

$$\textit{marital-status} \rightarrow \textit{capital-loss}$$

демонстрирует, что в большинстве случаев семейный статус прямо влияет на убытки. Эта информация может быть использована для дальнейшей аналитики.

Полученные результаты демонстрируют, что реализованный модуль профилирования успешно справляется с поставленной ему задачей и способен провести наукоёмкое профилирование, результаты которого могут показать пользователю новую информацию о данных.

# Заключение

В рамках данной работы были достигнуты следующие результаты:

1. Составлен перечень требований к модулю профилирования данных.
2. Разработана потенциальная архитектура модуля профилирования данных.
3. Реализован модуль профилирования данных.
4. Реализован набор тестов, покрывающий кодовую базу.
5. Проведена апробация, которая показала успешную применимость реализованного модуля.

Разработка приложения велась в личном репозитории[7] GitHub, который в дальнейшем был передан организации Desbordante.

[7] https://github.com/DaniilGoncharov/desbordante-profiler

# Список литературы


[1] Agrawal Rakesh, Srikant Ramakrishnan. Fast algorithms for mining association rules // Readings in Database Systems (3rd Ed.). — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1998. — P. 580–592. — ISBN: 1558605231.

[2] Bertrand Francois. Sweetviz: an open source Python library that generates beautiful, high density visualizations to kickstart EDA (Exploratory Data Analysis) with a single line of code. — https://github.com/fbdesignpro/sweetviz. — 2025.

[3] DataCleaner. — 2025. — URL: https://datacleaner.github.io/.

[4] Chernishev George, Polyntsov Michael, Chizhov Anton et al. Desbordante: from benchmarking suite to high-performance science-intensive data profiler (preprint). — 2023. — 2301.05965.

[5] Efficient Differential Dependency Discovery / Shulei Kuang, Honghui Yang, Zijing Tan, Shuai Ma // Proc. VLDB Endow. — 2024. — Vol. 17, no. 7. — P. 1552–1564. — URL: https://doi.org/10.14778/3654621.3654624.

[6] Rekatsinas Theodoros, Chu Xu, Ilyas Ihab F., Ré Christopher. HoloClean: Holistic Data Repairs with Probabilistic Inference. — 2017. — 1702.00820.

[7] IBM InfoSphere Information Server. — 2025. — URL: https://www.ibm.com/information-server.

[8] Informatica Data Quality. — 2025. — URL: https://www.informatica.com/products/data-quality.html.

[9] Jakob Wenzel, Rhinelander Jason, Moldovan Dean. pybind11 — Seamless operability between C++11 and Python. — https://github.com/pybind/pybind11. — 2025.

[10] Lux. Lux: A Python API for Intelligent Visual Discovery. — https://github.com/lux-org/lux. — 2025.

[11] Metanome. — 2025. — URL: https://hpi.de/naumann/projects/data-profiling-and-analytics/metanome-data-profiling.html.

[12] Redman Thomas C. Bad Data Costs the U.S. $3 Trillion Per Year // Harvard Business Review. — 2016. — URL: https://hbr.org/2016/09/bad-data-costs-the-u-s-3-trillion-per-year.

[13] Rydning David Reinsel-John Gantz-John, Reinsel John, Gantz John. The digitization of the world from edge to core // Framingham: International Data Corporation. — 2018. — Vol. 16. — P. 2–3.

[14] Song Shaoxu, Chen Lei. Discovering matching dependencies // Proceedings of the 18th ACM Conference on Information and Knowledge Management. — CIKM 09. — New York, NY, USA : Association for Computing Machinery, 2009. — P. 1421–1424. — URL: https://doi.org/10.1145/1645953.1646135.

[15] Talend. — 2025. — URL: https://www.talend.com.

[16] Tane: An Efficient Algorithm for Discovering Functional and Approximate Dependencies / Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, Hannu Toivonen // The Computer Journal. — 1999. — 01. — Vol. 42, no. 2. — P. 100–111. — https://academic.oup.com/comjnl/article-pdf/42/2/100/986909/420100.pdf.

[17] Wang Pei, He Yeye. Uni-Detect: A Unified Approach to Automated Error Detection in Tables // Proceedings of the 2019 International Conference on Management of Data. — SIGMOD '19. — New York, NY, USA : Association for Computing Machinery, 2019. — P. 811–828. — URL: https://doi.org/10.1145/3299869.3319855.

[18] YData. ydata-profiling: Data quality profiling & exploratory data analysis for Pandas and Spark DataFrames. — https://github.com/ydataai/ydata-profiling. — 2025.