Санкт-Петербургский государственный университет

Шангареев Рустам Ильшатович

Выпускная квалификационная работа

Разработка инструмента для автоматической предобработки временных рядов с адаптивными стратегиями

Уровень образования: бакалавриат

Направление *09.03.04 «Программная инженерия»*

Основная образовательная программа *СВ.5080.2021 «Программная инженерия»*

Научный руководитель:
ассистент кафедры системного программирования В.И. Гориховский

Рецензент:
Инженер ключевых проектов, ООО «МПГ АйТи Солюшнз» Р.С. Пусев

Санкт-Петербург
2025

Saint Petersburg State University

Rustam Shangareev

Bachelor's Thesis

Development of a tool for automatic preprocessing of time series with adaptive strategies

Education level: bachelor

Speciality *09.03.04 "Software Engineering"*

Programme *CB.5080.2021 "Software Engineering"*

Scientific supervisor:
Assistant, Candidate of Physical and Mathematical sciences V.I. Gorikhovskii

Reviewer:
Engineer of key projects at "IT Solutions", Candidate of Physical and Mathematical sciences
R.S. Pusev

Saint Petersburg
2025

Оглавление

Введение

Для решения различных задач статистики студентами и инженерами СПбГУ был создан проект PySATL [23]. Одна из его целей — реализация различных алгоритмов для работы с временными рядами, обеспечивая общий интерфейс для исследования найденных алгоритмов, их тестирования на реальных и синтетических данных. Не менее важной задачей является также автоматизация этих процессов.

На данный момент в проекте разрабатывается библиотека PySATL-CPD [24] для решения задачи обнаружения разладки [6], а также планируется внедрение новых библиотек для анализа временных рядов, например, для поиска аномалий [2]. Эффективность решения таких задач напрямую зависит от качества предобработки данных.

Во-первых, алгоритмы анализа временных рядов разделяются на онлайн, то есть работающие в реальном времени и имеющие информацию для своего анализа только на конкретный момент времени t и до него, и оффлайн — это алгоритмы, которые могут использовать для решения своей задачи весь временной ряд. Зачастую есть необходимость адаптировать какое-то оффлайн решение к онлайн, в этом может помочь предобработка: так, например, в PySATL-CPD [24] используется нарезка временного ряда в реальном времени скользящим окном для решения онлайн задачи обнаружения разладки с помощью оффлайн–алгоритма.

Во-вторых, реальные данные с временными метками часто имеют высокую размеренность и зашумлены. Первый пример таких данных — это данные о сделках с биржи: только лишь на 1 бирже может происходить по миллиону сделок каждую секунду, и если мы хотим проводить анализ, учитывать все сделки проблематично. Для решения этой проблемы придумали бары [5] — агрегированные по какому-то признаку данные о сделках (например, за каждые 5 минут) с расчетом различных статистик. Второй пример: данные измерительных приборов имеют систематическую погрешность, которую можно считать шумом, для ее устранения есть, например, фильтры Калмана [18] и Чебышева [7],

которые применяются к временным рядам

В-третьих, даже самая хорошая модель на необработанных данных будет работать плохо согласно концепту «garbage in, garbage out» [12].

Однако проведенный анализ существующих решений показал, что инструменты фокусируются на узкоспециализированных алгоритмах, оставляя вопросы конфигурируемой предобработки без системного решения.

Таким образом, насущной является задача предобработки временных рядов с адаптивными стратегиями для проекта PySATL.

1. Требования

В данной главе будут изложены требования к разрабатываемой системе для предобработки временных рядов, полученные от разработчиков PySATL [23], которые реализуют следующие библиотеки: PySATL-CPD, PySATL-experiments, PySATL-mptest, PySATL-criterion, PySATL-nmvm.

1.1. Нефункциональные требования

Нефункциональные требования для библиотеки оказались следующими.

1. **Обработка потоковых данных и ленивость вычислений.**
   Система должна обеспечивать поддержку обработки потенциально бесконечных временных рядов в режиме онлайн, формируя результат по мере поступления данных. Для этого все основные преобразования обязаны реализовываться с использованием ленивых вычислений, чтобы не требовать загрузки всей последовательности данных в память.

2. **Расширяемость.**
   Архитектура системы должна обеспечивать возможность расширения функциональности конечным пользователем. Пользователь должен иметь возможность реализовывать собственные обработчики, интегрировать их в конвейер обработки и дополнительно настраивать логику преобразования данных без необходимости модификации кода существующих компонентов.

3. **Интероперабельность.**
   Система должна быть совместима и легко интегрироваться с внешними библиотеками и программными средствами, используемыми для анализа временных рядов (например, для поиска аномалий, проведения кросс-валидации, визуализации).

4. **Надёжность и устойчивость к ошибкам.**
   Система должна обладать высокой надёжностью и быть устойчивой к нештатным ситуациям (например, пропуск данных, нестандартные форматы, ошибки потоков ввода-вывода). Продолжительная работа в потоковых сценариях не должна вызывать утечки памяти или прекращение работы компонентов. При возникновении ошибок должны использоваться явно определённые механизмы их обработки и информирования пользователя.

5. **Тестируемость.**
   Система должна строиться с ориентацией на покрытие кода модульными тестами. Все ключевые компоненты обязаны иметь независимые точки входа для проведения автоматического тестирования различных сценариев использования. Покрытие кода модульными тестами должно составлять не менее 95 %.

6. **Документированность и воспроизводимость.**
   Код системы должен сопровождаться полной технической документацией (docstrings, пользовательские гайды, примеры компоновки обработчиков), а результаты работы системы должны быть воспроизводимы при одинаковых входных данных и настройках.

1.2. Функциональные требования

Для инструмента были сформулированы также функциональные требования. Он должен иметь компоненты, перечисленные ниже.

1. Десериализаторы, предназначенные для десериализации данных из различных источников с конфигурируемый стратегией десериализации и возможностью их работы в асинхронном и многопоточном режимах:

   - десериализатор для коллекций с численными данными Python и библиотеки numpy;
   - десериализатор для данных, поступаемых из файла;

- десериализатор для данных, поступаемых по WebSocket с конфигурируемой стратегией десериализации;
- десериализатор для данных, поступаемых из базы данных для различных СУБД.

2. Онлайн и оффлайн компоненты для кросс-валидации временных рядов с конфигуриуемым размером тренировочной и валидационной выборок: по временному ряду $\{X_i\}_{i=1}^n$ и введенным min_train_size $= m$ и val_size $= k$ компонент должен возвращать

$$(\text{Train}_t, \text{Val}_t)_{t=1}^T,$$

где

$$T = \begin{cases} \left\lfloor \dfrac{n-m}{k} \right\rfloor, & \textbf{для оффлайн задачи} \\ \left\lfloor \dfrac{t-m}{k} \right\rfloor, & \textbf{для онлайн задачи} \quad (\text{на момент времени } t) \end{cases},$$

$$\text{Train}_t = \{X_i \mid 1 \le i \le m + (t-1)k\},$$

$$\text{Val}_t = \{X_i \mid m + (t-1)k + 1 \le i \le m + tk\}.$$

3. Онлайн и оффлайн компоненты для скраббирования временных рядов скользящим окном с конфигурируемым размером окна и относительным размером сдвига: по временному ряду $\{X_i\}_{i=1}^n$ и длине окна window_length $= L$ и относительному размеру сдвига shift_factor $= \alpha$ компонент должен возвращать

$$\text{Window}_t = \{X_i \mid s_t \le i < s_t + L\},$$

где

$$s_t = (t-1) \cdot \lfloor \alpha \cdot L \rfloor.$$

4. Онлайн и оффлайн компоненты для сегментирования временных рядов с конфигурируемым правилом сегментации: по временному ряу $\{X_i\}_{i=1}^n$ и правилу сегментации $\text{Rule}(X, i)$ компонент должен

возвращать

$$\text{Segments} = \{\{X_i\}_{i=s_k}^{e_k}\}_{k=1}^{K},$$

где

$$s_1 = 1,\ e_K = n,$$

$$\forall k \in [1, K-1]: \begin{cases} s_{k+1} = e_k + 1 \\ e_k = \max\{j \mid \text{Rule}(X, j) = \text{True},\ s_k \leq j < n\} \end{cases}.$$

5. Онлайн и оффлайн компоненты для взятия выборки из временного ряда с конфигурируемым правилом выбора: по временному ряду $\{X_i\}_{i=1}^{n}$ и правилу отбора $\text{Select}(X, i)$ компонент должен возвращать

$$\text{Sample} = \{X_i \mid \text{Select}(X, i) = \text{True},\ 1 \leq i \leq n\}.$$

6. Онлайн и оффлайн компоненты для агрегации временного ряда с конфигурируемым правилом агрегации: по временному ряду $\{X_i\}_{i=1}^{n}$ и правилу агрегации $\text{Aggregate}(\cdot)$ компонент должен возвращать

$$\text{Aggregates} = \{\text{Aggregate}(\{X_i\}_{i=s_k}^{e_k})\}_{k=1}^{K},$$

где

(s_k, e_k) определяются правилом сегментации $\text{Rule}(X, i)$.

7. Онлайн компонент для сглаживания временного ряда с конфигурируемым фильтром. По временному ряду $\{X_i\}_{i=1}^{n}$, алгоритму фильтрации Filter (например, Калмановский фильтр) и конфигурации фильтра θ компонент должен возвращать

$$Z_i = \text{Filter}(X_1, X_2, \ldots, X_i;\ \theta).$$

8. Оффлайн компонент для фильтрации временного ряда с конфигурируемой функцией фильтрации: по временному ряду $\{X_i\}_{i=1}^{n}$, функции фильтрации $\text{Filter}(X, \varphi)$ и набору параметров φ компо-

нент должен возвращать

$$\{Z_i\}_{i=1}^n = \text{Filter}(\{X_i\}_{i=1}^n; \varphi).$$

9. Компонент для фильтра Калмана: по временному ряду наблюдений $\{z_t\}_{t=1}^n$ и начальным параметрам системы $(F, H, B, Q, R, x_0, P_0, u_t)$ возвращает последовательность оценок

$$\begin{cases} x_{t|t-1} = Fx_{t-1|t-1} + Bu_t \\ P_{t|t-1} = FP_{t-1|t-1}F^T + Q \\ K_t = P_{t|t-1}H^T(HP_{t|t-1}H^T + R)^{-1} \\ x_{t|t} = x_{t|t-1} + K_t(z_t - Hx_{t|t-1}) \\ P_{t|t} = (I - K_tH)P_{t|t-1} \end{cases}$$

10. Компонент для двойного экспоненциального сглаживания: по ряду $\{x_t\}_{t=1}^n$ и параметру сглаживания α возвращает последовательность

$$DEMA_t = 2s_t - s'_t, \qquad s_t = \alpha x_t + (1-\alpha)s_{t-1}, \quad s'_t = \alpha s_t + (1-\alpha)s'_{t-1}$$

с

$$s_1 = x_1, s'_1 = s_1.$$

11. Компонент для экспоненциального сглаживания: по ряду $\{x_t\}_{t=1}^n$ и параметру сглаживания α возвращает последовательность

$$y_t = \alpha x_t + (1-\alpha)y_{t-1}$$

с $y_1 = x_1$.

12. Компонент для сглаживания с весами Фибоначчи: по ряду $\{x_t\}_{t=1}^n$, окну размера N, где вес i-го члена равен F_i (число Фибоначчи), возвращает

$$y_t = \frac{\sum_{i=0}^{N-1} F_i x_{t-i}}{\sum_{i=0}^{N-1} F_i}$$

13. Компонент для Hull Moving Average: по ряду $\{x_t\}_{t=1}^n$, окну размера N возвращает последовательность

$$y_t = \frac{\sum_{i=0}^{M-1}(M-i)\left[2w_{t-i}^{\text{half}} - w_{t-i}^{\text{full}}\right]}{\sum_{i=1}^{M} i}$$

где $w_t^{\text{half}} = \frac{\sum_{j=0}^{N/2-1}(N/2-j)x_{t-j}}{\sum_{j=1}^{N/2} j}$, $w_t^{\text{full}} = \frac{\sum_{j=0}^{N-1}(N-j)x_{t-j}}{\sum_{j=1}^{N} j}$, $M = \lfloor\sqrt{N}\rfloor$.

14. Компонент для midpoint: по ряду $\{x_t\}_{t=1}^n$ и окну N возвращает

$$y_t = \frac{\max\{x_{t-N+1}, \ldots, x_t\} + \min\{x_{t-N+1}, \ldots, x_t\}}{2}$$

15. Компонент для midprice: по рядам $\{high_t\}_{t=1}^n$, $\{low_t\}_{t=1}^n$ и окну N возвращает

$$y_t = \frac{\max\{high_{t-N+1}, \ldots, high_t\} + \min\{low_{t-N+1}, \ldots, low_t\}}{2}$$

16. Компонент для OHLC4: по рядам $\{open_t\}_{t=1}^n$, $\{high_t\}_{t=1}^n$, $\{low_t\}_{t=1}^n$, $\{close_t\}_{t=1}^n$ возвращает

$$y_t = \frac{open_t + high_t + low_t + close_t}{4}$$

17. Компонент для PWMA: по ряду $\{x_t\}_{t=1}^n$, окну N, используя веса C_{N-1}^i, возвращает

$$y_t = \frac{\sum_{i=0}^{N-1} C_{N-1}^i x_{t-i}}{\sum_{i=0}^{N-1} C_{N-1}^i}$$

18. Компонент для RMA: по ряду $\{x_t\}_{t=1}^n$ и окну N возвращает

$$y_t = \frac{x_t + (N-1)y_{t-1}}{N}$$

с $y_1 = x_1$.

19. Компонент для SMA: по ряду $\{x_t\}_{t=1}^n$ и окну N возвращает

$$y_t = \frac{1}{N}\sum_{i=0}^{N-1} x_{t-i}$$

20. Компонент для T3 Moving Average: по ряду $\{x_t\}_{t=1}^n$, коэффициентам c_1, c_2, c_3, c_4, и значениям e_3, e_4, e_5, e_6 возвращает

$$y_t = c_1 e_6 + c_2 e_5 + c_3 e_4 + c_4 e_3$$

21. Компонент для TEMA: по ряду $\{x_t\}_{t=1}^n$ и трем скользящим экспоненциальным средним $s_t^{(1)}, s_t^{(2)}, s_t^{(3)}$ с одним окном возвращает

$$y_t = 3s_t^{(1)} - 3s_t^{(2)} + s_t^{(3)}$$

где

$$s_t^{(1)} = \alpha x_t + (1-\alpha) s_{t-1}^{(1)}, \quad s_t^{(2)} = \alpha s_t^{(1)} + (1-\alpha) s_{t-1}^{(2)}, \quad s_t^{(3)} = \alpha s_t^{(2)} + (1-\alpha) s_{t-1}^{(3)}.$$

22. Компонент для TRIMA: по ряду $\{x_t\}_{t=1}^n$ и окну N возвращает

$$y_t = \frac{1}{M} \sum_{j=0}^{M-1} \left(\frac{1}{M} \sum_{i=0}^{M-1} x_{t-j-i} \right)$$

где $M = \lceil N/2 \rceil$.

23. Компонент для VWAP: по рядам $\{volume_t\}_{t=1}^n$, $\{price_t\}_{t=1}^n$ и окну N возвращает

$$y_t = \frac{\sum_{i=0}^{N-1} volume_{t-i} \cdot price_{t-i}}{\sum_{i=0}^{N-1} volume_{t-i}}$$

24. Компонент для VWMA: по рядам $\{volume_t\}_{t=1}^n$, $\{price_t\}_{t=1}^n$ и окну N возвращает

$$y_t = \frac{\sum_{i=0}^{N-1} volume_{t-i} \cdot price_{t-i}}{\sum_{i=0}^{N-1} volume_{t-i}}$$

25. Компонент для WMA: по ряду $\{x_t\}_{t=1}^n$ и окну N возвращает

$$y_t = \frac{\sum_{i=1}^{N} i \, x_{t-N+i}}{\sum_{i=1}^{N} i}$$

26. Компонент для ZLMA: по ряду $\{x_t\}_{t=1}^{n}$ и окну N возвращает

$$y_t = m_t$$

где m_t — скользящее среднее типа MA по последовательности $\{2x_t - x_{t-lag}\}$, $lag = \lfloor (N-1)/2 \rfloor$.

27. Компонент для возможности конвейерной обработки временных рядов с помощью описанных выше компонентов.

2. Постановка задачи

Целью данной работы является разработка инструмента для автоматической предобработки временных рядов с адаптивными стратегиями. В связи с этим были поставлены задачи, перечисленные ниже.

1. Провести анализ требований к системе.
2. Провести общий обзор предметной области, проекта PySATL, существующих решений и используемых в них архитектур обработки временных рядов.
3. Разработать архитектуру инструмента для предобработки временных рядов с адаптивными стратегиями согласно функциональным требованиям.
4. Реализовать инструмент согласно разработанной архитектуре и требованиям.
5. Провести апробацию инструмента на библиотеке PySATL-CPD проекта PySATL.

3. Обзор

Настоящий обзор посвящён анализу существующих решений для предобработки временных рядов в контексте задач скраббирования, сегментации и агрегации. Структура исследования включает перечисленные ниже разделы.

- **Формализацию ключевых понятий** (Раздел 3.1): вводятся строгие определения онлайн/оффлайн анализа, сегментации, скрабирования и агрегации, а также их роль в конвейере обработки данных.
- **Сравнительный анализ существующих инструментов** (Раздел 3.2): проводится оценка функциональности решений по двум режимам работы (оффлайн/онлайн) с использованием кодовой системы классификации.
- **Критерии выбора технологий** (Раздел 3.3): обосновывается использование Python как базового языка для разработки.
- **Анализ проекта PySATL** (Раздел 3.4): рассматривается проект и существующая архитектура для обработки временных рядов.
- **Выводы** (Раздел 3.5): подводятся итоги обзорного анализа.

Цель исследования — выявить пробелы в существующих решениях и перенять полезные используемые практики в разрабатываемый инструмент.

3.1. Обзор задачи скраббирования

Анализ временных рядов — популярная область в машинном обучении: только лишь за 2024 год было опубликовано не менее 245 тысяч научных статей, имеющих упоминаний «time series analysis» [14]. Из примеров задач анализа временных рядов можно выделить: прогнозирование, обнаружение аномалий, обнаружение разладки, классификация, кластеризация. Введем более формальные определения.

Задача анализа временных рядов представляет собой класс проблем, направленных на извлечение информации из последовательностей данных, упорядоченных по временной оси. В общем случае задача формулируется как преобразование входного временного ряда $\{X_t\}_{t=1}^{T}$, где $X_t \in \mathbb{R}^d$, в некоторый выход (чаще всего метки, оценки или прогнозы), отражающий скрытые свойства или целевые характеристики ряда.

Онлайн-анализ решает задачу в режиме реального времени, обрабатывая данные последовательно по мере их поступления $\{X_t\}$ при $t \geq 1$. Для каждого момента времени t вывод Y_t генерируется на основе ограниченного окна наблюдений $\{X_{t-k}, \ldots, X_t\}$ (где k может быть фиксированным или адаптивным) с учетом вычислительных ограничений, требований к задержке, необходимости адаптации к изменяющимся условиям.

Оффлайн-анализ / анализ в автономном режиме / автономный анализ предполагает обработку полного исторического набора данных $\{X_t\}$ за фиксированный временной интервал $[1, T]$, где решение формируется с учетом всей доступной информации. Такой режим ориентирован на максимизацию точности выводов, применение вычислительно сложных методов, итеративную оптимизацию параметров, ретроспективную валидацию результатов.

Хоть и многие алгоритмы для решения задач на временных рядах способны находить закономерности и в никак не обработанном потоке данных, существуют и те, что требуют предобработку измерений, поступающих на вход. Например, при прогнозировании движения цены часто используют не всю информацию о сделках, а лишь сагрегированные в «бары» [5]. Кроме того, существуют исследования по специальной предобработке временных рядов, которые могут облегчить решение профильной задачи [8]. Отдельно стоит отметить адаптацию оффлайн-алгоритмов к онлайн-анализу: в проекте PySATL-CPD [24] для обнаружения разладки в реальном времени требуется «оконная» обработка

временного ряда. Введем используемые в этой работе связанные с предобработкой понятия.

Сегментацией временного ряда $\{y_i\}_{i=1}^n$ называется преобразование его к последовательности $\{(y_{f_i}, y_{f_i+1}, \ldots, y_{s_i})\}_{i=1}^k$, где

$$\forall j: \ 1 \leq j < k: \ s_j + 1 = f_{j+1}$$

$$1 = f_1 \leq s_1 < f_2 \leq s_2 < \cdots < f_{k-1} \leq s_{k-1} < f_k \leq s_k = n$$

Скраббированием временного ряда $\{y_i\}_{i=1}^n$ называется преобразование его к последовательности $\{\{y_x\}_{x\in[s_i,e_i]}\}_{i=1}^k$, где

$$\bigcup_{i=1}^{k}[s_i, e_i] \subset [1:n],$$
$$\forall\ i,j \in [1:k] : i < j \Rightarrow (s_i < s_j) \text{ или } (e_i < e_j).$$

Агрегацией временного ряда $\{y_i\}_{i=1}^n$ называется его скраббирование к последовательности $\{\{y_x\}_{x\in[s_i,e_i]}\}_{i=1}^k$, а затем применения некоторой агрегирующей функции f, что в результате дает

$$\{f\ (\{y_x\}_{x\in[s_i,e_i]})\}_{i=1}^k, \text{ где}$$

$$\bigcup_{i=1}^{k}[s_i, e_i] \subset [1:n],$$
$$\forall\ i,j \in [1:k] : i < j \Rightarrow (s_i < s_j) \text{ или } (e_i < e_j).$$

Таким образом, ниже резюмируется разница между видами предобработки.

- **Сегментация** помогает разбить задачу на подзадачи, например, в простейшем случае это может быть разбиение на окна фиксированного размера.

- **Скраббирование** похоже на сегментацию, но дополнительно позволяет исключать некоторые наблюдения из временного ряда, что может быть полезно, например, при наличии шума в наблюдениях.

- **Агрегация** же идет еще дальше, предоставляя функциональность преобразования сегментов временного ряда, например, применение функции среднего к небольшим одинаковым сегментам наблюдений «сглаживает» их значения. Также агрегация помогает снижать размерность временных рядов, выделяя основные статистики.

Для более наглядного представления разницы между видами предобработки рассмотрим их для одного случайного численного временного ряда на рис. 1.

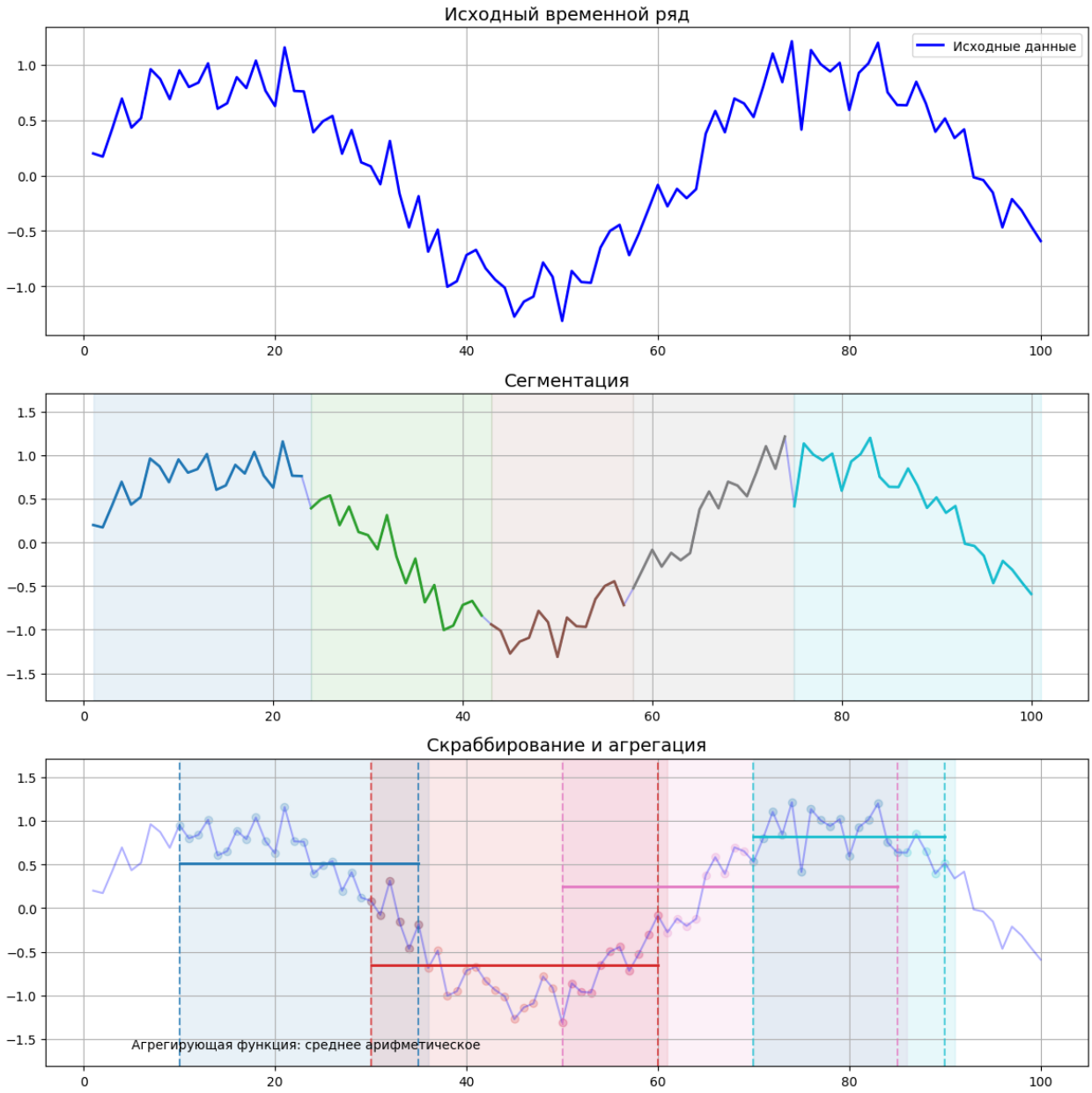

Рис. 1: Наглядная демонстрация разницы между видами предобработки.

3.2. Обзор существующих решений

Для исследования существующих решений был произведён поиск библиотек, предназначенных для анализа временных рядов, с помощью запросов в Google, Google Scholar, GitHub и Gitlab: «time series aggregation», «time series analysis», «data aggregation tool», «time series library», «time series segmentation». Был составлен следующий список из существующих инструментов, относящихся к различным областям исследований.

- Palantir [22] – проприетарная платформа с закрытым исходным кодом. Предоставляет возможности для онлайн и оффлайн агрегации через кастомные обёртки, однако основной акцент сделан на графический интерфейс.
- Elastic [9] – проприетарное решение с открытым форком. Документация не позволяет однозначно оценить возможности сложной агрегации.
- Apache Spark [3], PipelineDB [29] – решения для распределённых баз данных с открытым исходным кодом. Обеспечивают эффективное хранение потоковых данных и базовые оконные агрегации.
- Mathis Wellmann projects [32] – набор библиотек на Rust и Go для агрегации торговых данных со стандартными правилами отбора и возможностью расширения функциональности.
- Ta4J [28] – Java-библиотека для оффлайн агрегации финансовых временных рядов с фиксированными правилами отбора.
- PandasTA [30] – Python-библиотека, использующая стандартные методы pandas (groupby, resample) для базовой агрегации.
- Tsam [10] – решение для кластеризации временных рядов методами k-средних, иерархической кластеризации и усреднения. Сегментация используется только для учёта сезонности.

- ClasPy [27] – библиотека для обнаружения точек разладки через бинарную классификацию с использованием k-NN и статистических тестов.

- Skforecast [26] – инструмент для прогнозирования временных рядов с использованием методов машинного обучения, включая обнаружение аномалий.

- Time series aggregation in R [13] – экосистема R содержит библиотеку для сегментации

- Julia [17] – пакет Julia для агрегации с настраиваемыми функциями внутри предопределённых сегментов.

- TsDownsample [16] – высокопроизводительное решение для понижения размерности временных рядов через равномерное сэмплирование и простые алгоритмы сжатия.

- PATSS [31] – алгоритм семантической сегментации на основе Symbolic Aggregate Approximation (SAX) с преобразованием в эмбеддинги.

- PyTS [11] – библиотека преобразований временных рядов, включая Piecewise Aggregate Approximation (PAA) и методы кластеризации.

- MatrixProfile [8] – архивный проект с реализацией алгоритмов Matrix Profile для обнаружения аномалий и точек разладки.

- NeuroKit2 [21] – Python-библиотека для анализа медицинских временных рядов (ЭКГ, ЭЭГ, дыхание) с инструментами предобработки, извлечения признаков и статистического анализа.

- HeartPy [15] – специализированное решение для обработки данных сердечного ритма с акцентом на анализ сигналов носимых устройств, включая обнаружение пиков и расчёт вариабельности.

- TA-Lib [19] – эталонная библиотека технического анализа финансовых данных с поддержкой 150+ индикаторов (ие средние, RSI, MACD) и свечных паттернов.

- QuantLib [1] – комплексный фреймворк для количественных финансов с продвинутыми моделями ценообразования, управления рисками и обработки временных рядов.

- Kats [25] – библиотека от Meta для инженерного анализа временных рядов, предлагающая инструменты прогнозирования, обнаружения аномалий и анализа трендов.

- MetPy [20] – метеорологический пакет для анализа атмосферных временных рядов с поддержкой специализированных расчётов (градиенты, адвекция) и визуализации.

- Flow Forecast [4] – фреймворк для прогнозирования транспортных потоков на базе PyTorch с предконфигурированными архитектурами глубокого обучения.

3.2.1. Сравнение существующих решений по функциональности предобработки

Далее сравним эти инструменты и библиотеки между собой по различным видам функциональности, которые могут помочь в решении оригинальной задачи реализации скрабберов. Предварительно также определим легенду кодовых вердиктов по анализу функциональности инструмента.

Таблица 1: Легенда кодовых вердиктов

Цвет	**Код**	**Описание**
	A: Базовая агрегация	Простая группировка с константными правилами. Это агрегация временных рядов (или финансовых данных [5]) с отбором по константому значению признака.
	B: Код научных статей	Реализация методов предобработки, описанных в научных публикациях
	C: Отсутствие функциональности	Отсутствие встроенной поддержки методов предобработки

Таблица 2: Сводная таблица функциональности инструментов

Инструмент	**Оффлайн**	**Онлайн**
Palantir	A	A
Elastic	A	C
Apache Spark	A	C
PipelineDB	A	C
Wellmann projects	A	A
Ta4J	A	A
PandasTA	A	C
Tsam	A	C
ClasPy	C	C
Skforecast	C	C
TSA in R	A	C
Julia	A	C
TsDownsample	A	C
PATSS	B	C
PyTS	A	C
MatrixProfile	A	C
NeuroKit2	C	C
HeartPy	C	C
TA-Lib	A	C
QuantLib	C	C
Kats	C	C
MetPy	C	C
Flow Forecast	C	C

Таким образом, даже оффлайн задачу решает небольшое количество инструментов, которые при этом предоставляют только базовую функциональность, не представляющую особую ценность. По сравнению с оффлайн задачей, онлайн задачу решает лишь 1 проприетарная платформа, а также 2 библиотеки для работы с финансовыми данными и тривиальными правилами агрегации. Более того, нет ни одного существующего решения с архитектурой онлайн и оффлайн предобработки временных рядов.

3.3. Обзор используемых технологий

Язык Python был выбран в качестве языка программирования для выполнения задач по следующим причинам.

- **Язык проекта PySATL [23]**: на 26.03.2025 на Python написано 100 % кода проекта.
- **Распространенность в академической среде**: из рассмотренных в обзоре библиотек и инструментов лишь 2 библиотеки имеют открытый код, написанный не на Python, при этом многие из библиотек на Python могут быть полезны для работы с временными рядами. Например, PATSS [31] предлагает новый метод агрегации, написанный на Python, который можно переиспользовать.
- **Интероперабельность с языками, ориентированными на высокопроизводительные вычисления**: в случае необходимости медленная функциональность может быть написана на более производительных языках программирования (C++/C, Rust), что достигается поддержкой нативных биндингов в Python.

3.4. PySATL

Для решения различных задач статистики был создан проект PySATL [23]. Одна из целей его разработчиков — реализация с общим интерфейсом различных алгоритмов для временных рядов. Не менее важной задачей проекта является исследование найденных алгоритмов, их тестирование на реальных и синтетических данных, а также автоматизация этих процессов.

На данный момент в проекте разрабатывается библиотека PySATL-CPD [24] для решения оффлайн и онлайн задачи обнаружения разладки, а также планируется внедрение новых библиотек для анализа временных рядов, например, для поиска аномалий, что было сказано в личной беседе с разработчиками.

Рассмотрим упрощённую схему архитектуры проекта PySATL-CPD, в которой уже реализована некоторая логика скраббирования для задачи обнаружения разладки.

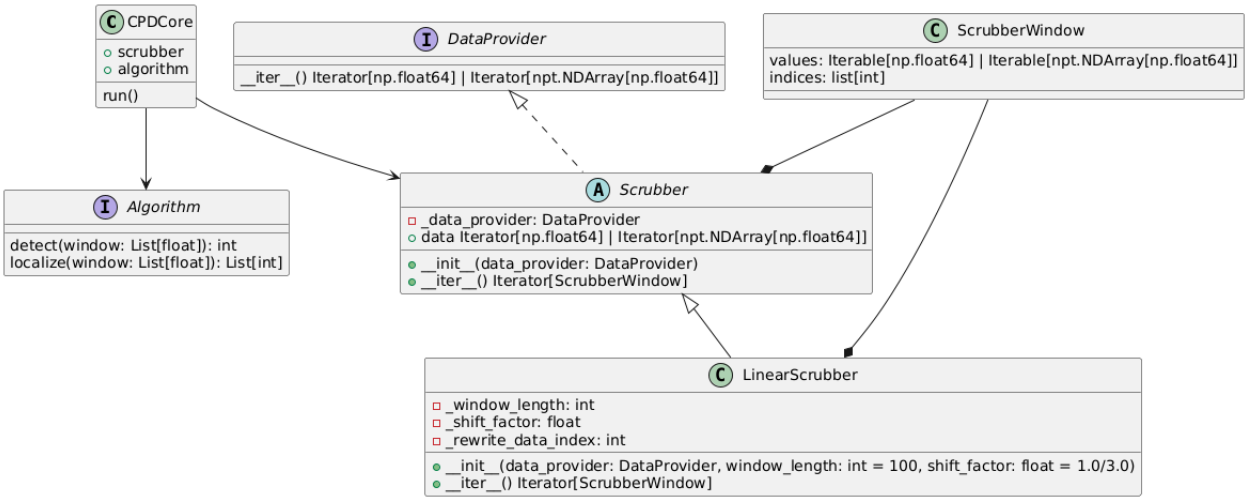

Рис. 2: Часть архитектуры пакета PySATL-CPD на 26.03.2025.

3.5. Выводы

Проведённый обзор показал, что существующие решения для анализа временных рядов в основном ориентированы на узкоспециализированные задачи (прогнозирование, поиск аномалий и т.д.) и предлагают ограниченную функциональность для агрегации данных. Большинство инструментов поддерживают лишь простейшие правила предобработки, особенно в онлайн-режиме, где доминируют проприетарные платформы. При этом отсутствуют библиотеки, совмещающие гибкую архитектуру для онлайн и оффлайн обработки с возможностью настройки сложных правил скрабирования. Это создаёт предпосылки и дополнительные требования для разработки нового инструмента.

1. Требование универсального итераторного API: реализована возможность любого пайплайна обработки быть прозрачным итерируемым объектом Python, что позволяет использовать предобработанные данные ”на лету” в сторонних инструментах без промежуточной материализации.

2. Возможность конструирования пайплайнов обработки данных при помощи операторов композиции (например, конструкция вида `handler1 | handler2 | handler3`), что повышает удобство и выразительность описания последовательности операций.

3. Весь код библиотеки ориентирован поддержку частичной обработки данных: результат любого шага либо может быть недоступен (например, возвращается `None`), либо поступает на следующий этап только при готовности — таким образом, возможна работа с сильно разреженными, неравномерными или неполными рядами.

4. Архитектура библиотеки предполагает простое внедрение пользовательских стратегий агрегации, фильтрации и сегментации не только через наследование, но и через передачу функций (callable-объектов) без обязательного создания новых классов.

5. Тестовое покрытие включает не только unit-тесты отдельных обработчиков, но и интеграционные сценарии для «длинных» пайплайнов с замыканием данных из разных источников и агрегаций, что гарантирует работоспособность библиотеки при интеграции с внешними решениями в течение долгого времени.

6. Компоненты библиотеки, если это возможно, допускает асинхронную или многопоточную работу, что актуально для интеграции с потоками в реальном времени и ускоряет обработку при больших нагрузках.

7. Модульные тесты для компонентов, которые реализованы в библиотеке PandasTA [30], должны иметь в качестве ожидаемого значения результат работы соответствующей функции в библиотеке PandasTA.

4. Архитектура

В этой главе представлена архитектура системы — инструмента для автоматической предобработки временных рядов с адаптивными стратегиями, ключевое назначение которого устранение типовых сложностей, возникающих при подготовке временных рядов для анализа и машинного обучения, таких как нестандартные форматы данных, неоднородность временных интервалов, необходимость кросс-валидации с учётом временной зависимости, а также адаптация методов обработки под динамически изменяющиеся условия.

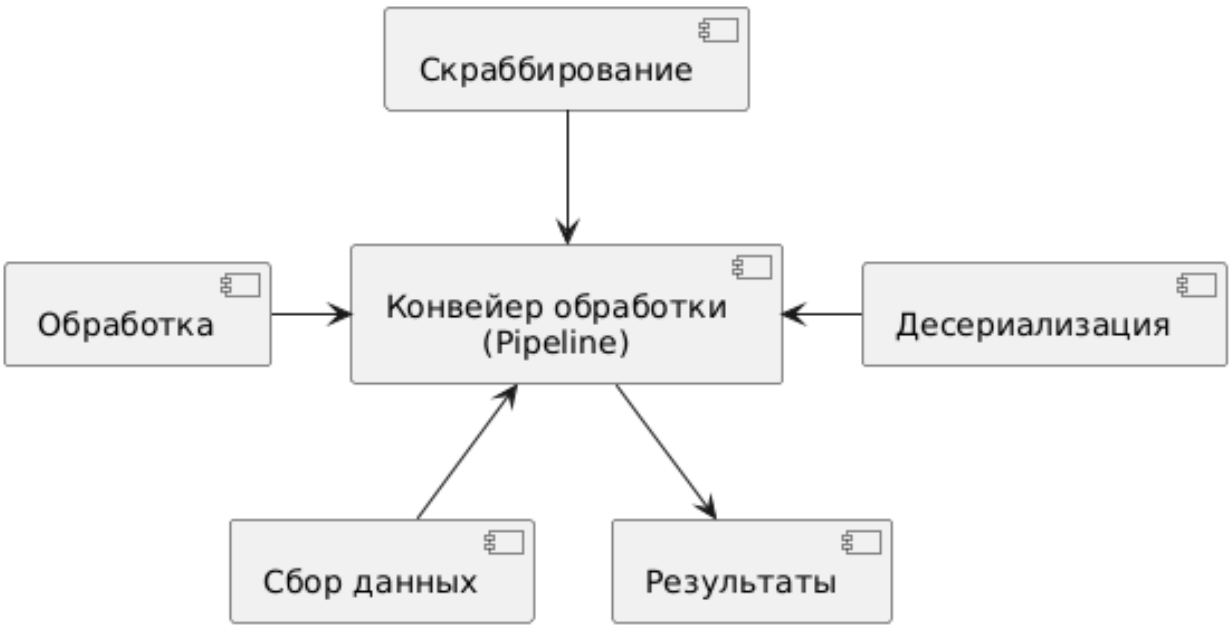

Рис. 3: Общая архитектура системы

На рисунке 3 представлена диаграмма компонент общей архитектуры решения, построенной на основе собранных требований к системе.

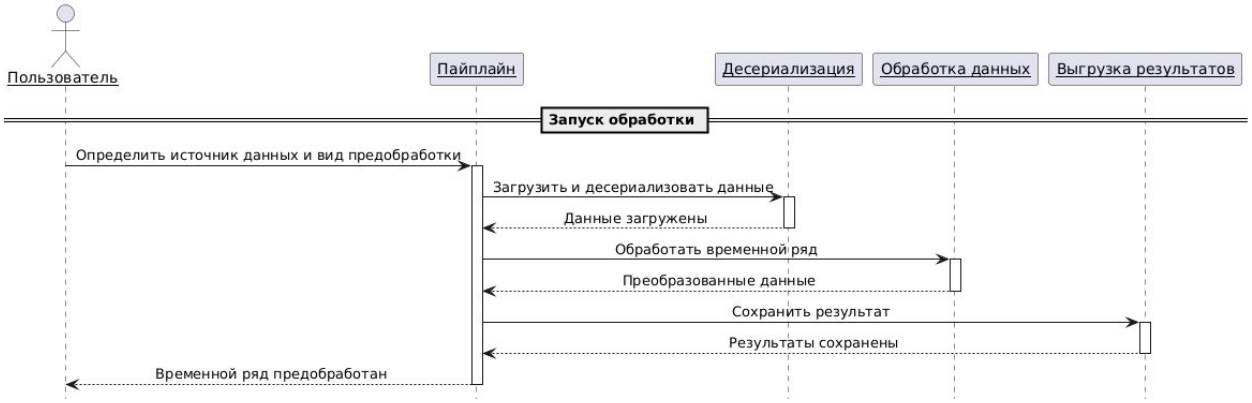

Рис. 4: Главный сценарий работы системы

На рисунке 4 представлена диаграмма главного сценария работы системы, состоящая из конвейерной загрузки и десериализации, обработки, преобразования и сохранения результата.

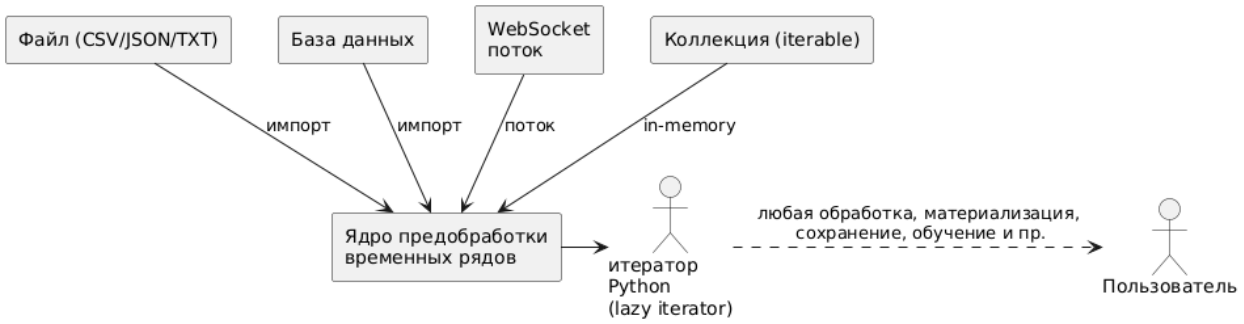

Рис. 5: Контекстная диаграмма: входные и выходные данные системы

Система предоставляет универсальный интерфейс, диаграмма которого показана на рисунке 3: все источники данных (файлы, базы данных, сетевые потоки) приводятся к единому внутреннему представлению, а результат работы ядра системы — это ленивый Python-итератор с элементами временного ряда. Такой подход позволяет

- эффективно работать с большими и/или неопределёнными по длине потоками данных;
- передавать результат обработки напрямую в последующие конвейеры, модули или итерировать над ним частично;

- отдавать пользователю полный контроль над объёмом и форматом дальнейшей обработки (например, преобразовать только часть последовательности или материализовать результат в удобную для задачи структуру).

В качестве инструментария было достаточно технологий, используемых в проекте PySATL [23]:

1. язык программирования: Python;
2. управление зависимостями: Poetry;
3. библиотека для анализа типов: MyPy;
4. библиотеки для тестирования компонентов: Pytest, pytest-mock, hypothesis.
5. библиотеки для оценки покрытия модульным тестированием компонентов: Pytest-cov.

5. Реализация

В данной главе будут описаны детали реализации и структуры системы, а также будут показаны диаграммы компонентов и их описания.

5.1. Верхнеуровневая архитектура

Для имплементации библиотеки были выделены 2 верхнеуровневых компонента. Далее они будут разобраны более подробно, сейчас же ниже будет представлено краткое описание.

- **TSP-Core** для хранения основной части библиотеки, состоящей из базовых/абстрактных классов, которые предполагают их расширение для использования, и поставщиков данных («DataProviders») для различных источников.
- **Implementations** — конкретные реализации и расширеня базовой библиотеки.

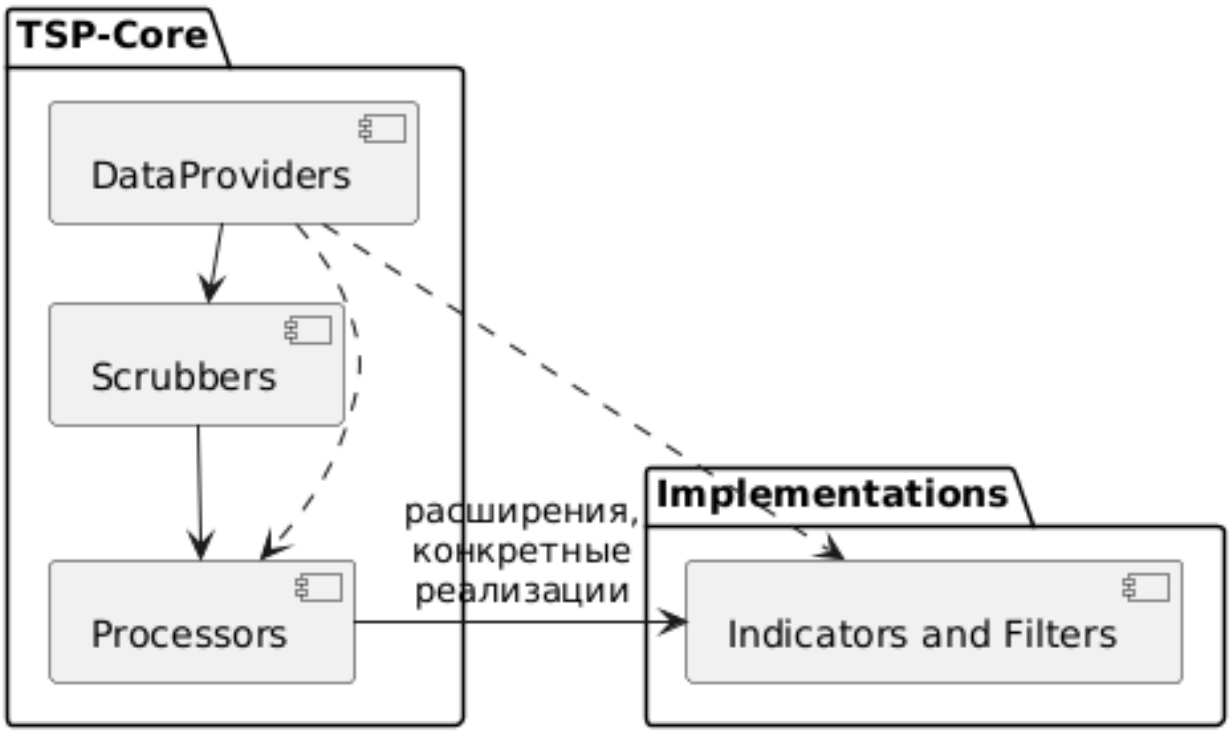

Рис. 6: Диаграмма верхнеуровневой архитектуры инструмента.

5.2. Компонент TSP-Core

Основной компонент («TSP-Core») был реализован с помощью следующих паттернов проектирования.

- Паттерн «Цепочка ответственности» в классе «Pipeline»
  - позволил организовать последовательную обработку данных через цепочку обработчиков;
  - дал возможность каждому обработчику отвечать за конкретный этап преобразования временного ряда;
  - упростил добавление/удаление этапов обработки без изменения клиентского кода.
- Паттерн «Итератор» в базовом абстрактном классе «Handler»
  - позволил лениво обрабатывать данные;
  - предоставил единый интерфейс для работы с разными источниками данных;
  - скрыл внутреннюю структуру данных от клиентского кода.
- Паттерн «Адаптер» в «DataBaseDataProvider»
  - дал возможность поставлять данные из различных СУБД с различными интерфейсами.

В остальном «TSP-Core» состоит из конкретных реализаций «DataProvider» для поставки данных, «Scrubber» для скраббирования временного ряда и «ScrubberWindow» для специального хранения конечно окна временного ряда в виде значений и индексов.

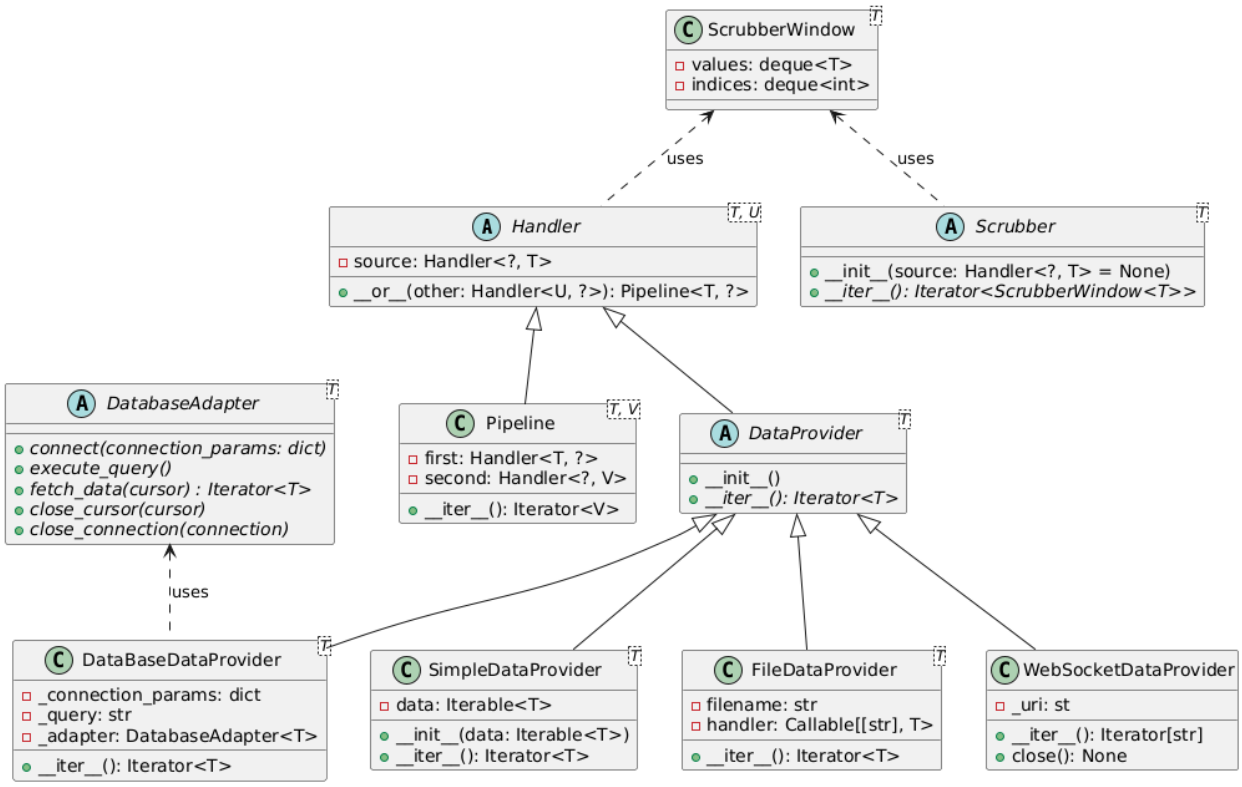

Рис. 7: Диаграмма основного компонента библиотеки («TSP-Core»).

Такая реализация позволяет использовать код, представленный на листинге 1.

Листинг 1: Код, который преобразовывает временной ряд «data» скользящим окном с агрегацией в другой временной ряд с помощью конвейерной обработки.

```
scrubber =
    SimpleDataProvider(data) |
    LinearScrubber(window_length=3) |
    MappingHandler(map_func=lambda x: sum(x) / len(x))
```

5.3. Компонент TSP-Core.processor

Компонент `tsp-core.processor` содержит в себе множество функциональностей, перечисленных ниже.

- `MappingHandler` – обработчик, который применяет к каждому

элементу последовательности заданную функцию преобразования (аналог map в функциональном программировании).

- `CombineHandler` – позволяет параллельно применять несколько обработчиков к одному источнику данных и агрегировать их вывод произвольной функцией объединения (pattern: fan-out/fan-in).

- `TeeHandler` – разветвляет поток данных на два пути: один обрабатывает данные дополнительным обработчиком, другой передаёт их в неизменном виде; затем результаты обеих веток совмещаются с помощью пользовательской функции.

- `OnlineFilterHandler`– реализует онлайн-фильтрацию: результат на каждом шаге вычисляется на основе набегающего окна истории с помощью заданной функции фильтра, что даёт возможность поточной обработки и адаптации к изменяющимся данным.

- `OfflineFilterHandler` – применяет фильтр к полному набору данных целиком (в пакетном режиме), предоставляя возможность использовать фильтры, требующие глобального контекста (например, спектральные преобразования, сглаживания "в одной пачке").

- `InductiveHandler` – абстрактный базовый класс для обработчиков, вычисляющих результат индуктивно: каждое новое значение вычисляется по предыдущему состоянию и текущему элементу последовательности (например, скользящие средние, суммы, экспоненциальные фильтры и др.).

- `LagHandler` – специальный обработчик, обеспечивающий задержку (лаг) потока данных на заданное количество шагов, что позволяет, например, реализовать операции сравнения текущего и прошлого значения.

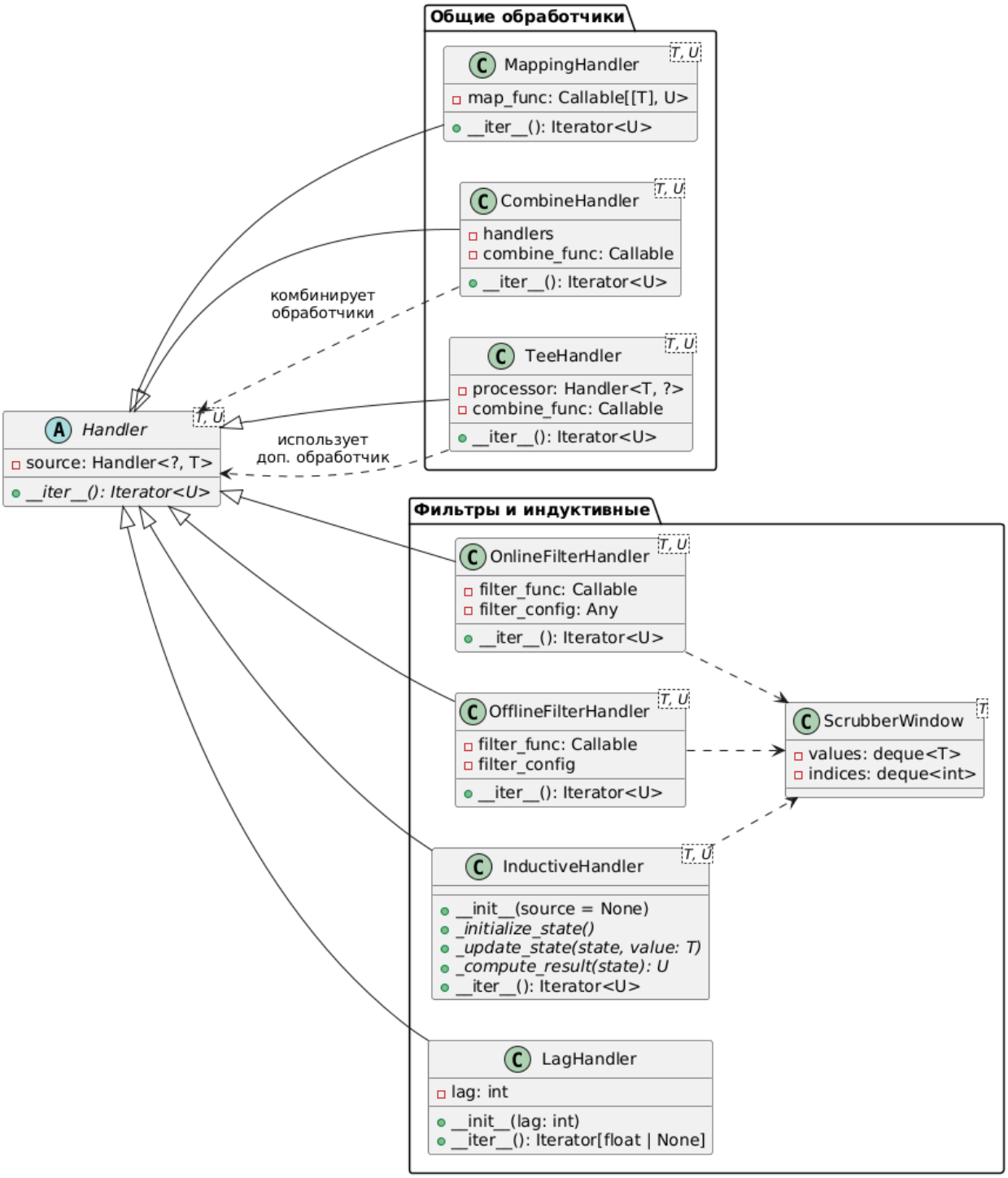

Рис. 8: Диаграмма классов компонента core.processor.

5.4. Компонент Implementations

Кроме расширяемого ядра библиотеки для реализации пользовательских обработчиков, были реализованы следующие преобразователи временных рядов.

1. *Kalman time series filter*: Фильтр Калмана для сглаживания временных рядов.
   Основан на рекурсивной оценке состояния x_k системы:

$$x_{k|k-1} = Fx_{k-1|k-1} + Bu_k, \qquad P_{k|k-1} = FP_{k-1|k-1}F^T + Q$$

   Оценка и коррекция по измерениям z_k:

$$K_k = P_{k|k-1}H^T(HP_{k|k-1}H^T + R)^{-1}$$

$$x_{k|k} = x_{k|k-1} + K_k(z_k - Hx_{k|k-1})$$

2. *Time series cross validator*: Кросс-валидация для временных рядов с учётом их структуры.
   Делит временной ряд на последовательные непересекающиеся окна (expanding window или rolling window). На каждом шаге обучение производится на $[x_1, \ldots, x_N]$, проверка на $[x_{N+1}, \ldots, x_{N+k}]$.

3. *Double Exponential Moving Average (DEMA)*: Двойное экспоненциальное сглаживание.
   Определяется как $DEMA = 2 \cdot EMA - EMA(EMA)$.

4. *Exponential Moving Average (EMA)*: Классическое экспоненциальное сглаживание.

$$EMA_t = \alpha x_t + (1 - \alpha)EMA_{t-1}, \quad \alpha = \frac{2}{N+1}$$

   где N — длина окна.

5. *Fibonacci's Weighted Moving Average (FWMA)*: Сглаживание с весами из последовательности Фибоначчи.

$$FWMA_t = \frac{\sum_{i=0}^{N-1} F_i x_{t-i}}{\sum_{i=0}^{N-1} F_i}$$

   где F_i — i-е число Фибоначчи.

6. *Hull Exponential Moving Average (HMA)*: Сглаживание с уменьшением запаздывания.

$$HMA_t = WMA\left(2 \cdot WMA(x, N/2) - WMA(x, N), \sqrt{N}\right)$$

где WMA — взвешенное скользящее среднее.

7. *Midpoint*: Сглаживание по среднему между максимумом и минимумом в окне.

$$Midpoint_t = \frac{\max\limits_{i=0..N-1} x_{t-i} + \min\limits_{i=0..N-1} x_{t-i}}{2}$$

8. *Midprice*: Сглаживание средним между максимальной "high" и минимальной "low" ценой за период.

$$Midprice_t = \frac{\max\limits_{i=0..N-1} high_{t-i} + \min\limits_{i=0..N-1} low_{t-i}}{2}$$

9. *Open-High-Low-Close Average (OHLC4)*: Сглаживание по среднему арифметическому OHLC-бара.

$$OHLC4_t = \frac{open_t + high_t + low_t + close_t}{4}$$

10. *Pascal's Weighted Moving Average (PWMA)*: Сглаживание с весами из треугольника Паскаля.

$$PWMA_t = \frac{\sum_{i=0}^{N-1} C_{N-1}^i x_{t-i}}{\sum_{i=0}^{N-1} C_{N-1}^i}$$

где C_{N-1}^i — биномиальные коэффициенты (треугольник Паскаля).

11. *Wilder's Moving Average (RMA)*: Скользящее среднее с коэффициентом $\alpha = 1/N$.

$$RMA_t = \frac{x_t + (N-1) \cdot RMA_{t-1}}{N}$$

где N — период сглаживания.

12. *Simple Moving Average (SMA)*: Простое скользящее среднее.

$$SMA_t = \frac{1}{N} \sum_{i=0}^{N-1} x_{t-i}$$

13. *T3 Moving Average*: Тройное экспоненциальное сглаживание по формуле Тимма Тиллсона.

$$T3_t = c_1 e_6 + c_2 e_5 + c_3 e_4 + c_4 e_3$$

где e_3–e_6 — последовательные EMA, $c_1, \ldots, c_4$ — специальные коэффициенты.

14. *Triple Exponential Moving Average (TEMA)*: Тройное экспоненциальное сглаживание.

$$TEMA_t = 3 \cdot EMA_1 - 3 \cdot EMA_2 + EMA_3$$

где EMA_1 — от исходных данных, EMA_2 — от EMA_1, EMA_3 — от EMA_2.

15. *Triangular Moving Average (TRIMA)*: Двойное сглаживание, применённое последовательно.
Получается как SMA от SMA: $TRIMA = SMA(SMA(x, \lceil \frac{n}{2} \rceil), \lceil \frac{n}{2} \rceil)$.

16. *Volume Weighted Average Price (VWAP)*: Средневзвешенная цена по объёму.

$$VWAP_t = \frac{\sum_{i=0}^{N-1} volume_{t-i} \cdot price_{t-i}}{\sum_{i=0}^{N-1} volume_{t-i}}$$

17. *Volume Weighted Moving Average (VWMA)*: Скользящее среднее с учётом объёма торгов.

$$VWMA_t = \frac{\sum_{i=0}^{N-1} volume_{t-i} \cdot price_{t-i}}{\sum_{i=0}^{N-1} volume_{t-i}}$$

(идентично предыдущему, если считать "price" = "close").

18. *Weighted Moving Average (WMA)*: Взвешенное скользящее среднее

с линейно изменяющимися весами.

$$WMA_t = \frac{\sum_{i=1}^{N} w_i x_{t-N+i}}{\sum_{i=1}^{N} w_i}$$

где веса w_i возрастают линейно: $w_i = i$.

19. *Zero Lag Moving Average (ZLMA)*: Сглаживание с компенсацией лага:

$$ZLMA_t = MA(2x_t - x_{t-lag})$$

где $lag = \lfloor (N-1)/2 \rfloor$, MA — любое среднее (по умолчанию EMA).

5.5. Стратегии расширяемости и повторного использования

В проекте реализованы методики проектирования, обеспечивающие гибкость архитектуры, повторное использование кода и возможность расширять функциональность без изменения основного кода библиотеки.

Базовые принципы будут перечислены ниже.

- Вся обработка строится вокруг абстрактных классов:
  - `Handler` — абстрактный базовый класс для всех обработчиков;
  - `DataProvider` — абстракция для источников данных;
  - `Scrubber` — абстракция для “оконных” обработчиков;
  - `DatabaseAdapter` — интерфейс для работы с различными СУБД.

  Это позволяет создавать свои производные классы, реализуя только часть интерфейса, необходимую для конкретной задачи.

- Шаблоны проектирования «Цепочка ответственности» (Chain of Responsibility) и «Пайплайн» позволяют связывать обработчики

через оператор пайпа (`|`), формируя произвольную цепочку трансформаций данных. Класс `Pipeline` инкапсулирует соединение обработчиков, позволяя унифицировано работать с длинными цепочками.

- Расширяемость через абстрактные точки (hooks) дает возможность реализации только метода `__iter__()` (или нужные «hooks» для обработки одного элемента/окна). Например, для индуктивных обработчиков (наследников `InductiveHandler`) нужно реализовать методы инициализации и обновления состояния, что позволяет создавать новые алгоритмы обработки последовательностей без дублирования логики управления потоком.

Классы, специально ориентированные на расширение и гибкость.

- `InductiveHandler` позволяет реализовывать алгоритмы, вычисляющие результат на основе предыдущего состояния (например, скользящие средние, накопительные суммы и т.д.). Для расширения достаточно реализовать методы `initialize_state`, `update_state`, `compute_result`.

- `CombineHandler`: даёт пользователю возможность параллельно применять несколько обработчиков и агрегировать их результаты произвольной функцией. Это облегчает комбинирование пайплайнов без переписывания существующих обработчиков.

- `TeeHandler`: позволяет «раздваивать» поток данных, применяя к одной ветви дополнительную обработку и объединяя результаты с помощью пользовательской функции.

- `Scrubber` и наследники: предоставляют универсальные средства разбиения потока на окна или сегменты. Пользователь может реализовать собственную стратегию разбиения, создав соответствующего наследника.

- `Handler`, `DataProvider` и остальные интерфейсные абстракции: позволяют создавать пользовательские источники данных и обработчики, реализующие только нужную логику `__iter__()` и легко интегрируются в любые пайплайны.

- `DatabaseAdapter`: обеспечивает расширение поддержки новых СУБД через реализацию стандартных методов подключения и получения данных.

Применение абстракций, поддержка композиции, система наследования обработчиков, а также наличие компонентов делают библиотеку удобной для дальнейшего развития и интеграции новых методов без необходимости модифицировать существующий код.

6. Апробация

В качестве апробации был проведено интеграционное тестирование и исследование совместной работы с модулем PySATL-CPD [24]. В данном разделе описаны цель, исследовательские вопросы и условия экспериментального сравнения, а также продемонстрированы результаты и выводы из них.

6.1. Цель и вопросы эксперимента

Целью эксперимента является определение влияния различных предобработок временных рядов на качество решения задачи об обнаружении разладки. Для достижения цели были поставлены следующие исследовательские вопросы.

RQ1: Как меняются метрики качества решения задачи обнаружения разладки с предобработанным временным рядом по сравнению с необработанным на различных алгоритмах библиотеки PySATL-CPD [24]?

RQ2: Является ли сам инструмент и инструмент, интегрированный с другой библиотекой для анализа временных рядов, надежным и устойчивым к нештатным ситуациям?

6.2. Условия эксперимента

В данном разделе описывается тестовый стенд, на котором проводилось экспериментальное исследование.

6.2.1. Наборы данных

Эксперименты для оценки качества были проведены на синтетических данных размера 500 значений, среди которых находилась точка 250 с разладкой. Для генерации данных были использованы распределения:

- бета-распределение;
- равномерное распределение;

- нормальное распределение;
- распределение Вейбулла;
- экспоненциальное распределение.

6.2.2. Метрики оценки качества

Для комплексной оценки качества алгоритмов обнаружения разладок были выбраны следующие метрики:

1. **Задержка обнаружения (Detection Delay)** — среднее число отсчетов между реальным моментом разладки и моментом её обнаружения алгоритмом. Эта метрика позволяет оценить, насколько быстро алгоритм реагирует на изменение. Обозначим истинное положение разладки как t^*, а момент, в который алгоритм выдал сигнал, как $\hat{t}$. Тогда задержка обнаружения определяется как:

$$\text{Delay} = |\hat{t} - t^*|$$

   где усреднение производится по всем найденным разладкам.

2. **Процент обнаруженных разладок (Detection Rate, DR)** — отношение числа корректно найденных разладок к общему числу реально произошедших разладок. Эта метрика характеризует чувствительность алгоритма.

   Пусть N_{tp} — число корректно найденных разладок, N_{all} — число всех разладок:

$$\text{Detection Rate} = \frac{N_{tp}}{N_{all}} \cdot 100\%$$

3. **Частота ложных тревог (False Positive Rate, FPR)** — отношение числа ложноположительных срабатываний к общему числу отрицательных случаев (отсутствие разладки).

Пусть N_{fp} — число ложных тревог, N_{neg} — число реальных отрицательных случаев:

$$\text{FPR} = \frac{N_{fp}}{N_{neg}}$$

4. **Precision (точность)** — доля верно найденных разладок среди всех срабатываний алгоритма.

   Пусть N_{tp} — число истинно положительных срабатываний (корректно обнаруженных разладок), N_{pred} — общее число срабатываний:

$$\text{Precision} = \frac{N_{tp}}{N_{pred}}$$

5. **F1-мера** — гармоническое среднее между точностью (Precision) и полнотой (Recall), что позволяет сбалансированно учесть оба этих аспекта.

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

   где полнота (Recall) равна:

$$\text{Recall} = \frac{N_{tp}}{N_{all}}$$

Использование данных метрик обоснована тем, что задача обнаружения разладок критична к как к пропуску изменений, так и к ложной сигнализации, что определяет необходимость оценки как чувствительности, так и специфичности алгоритмов. Метрика задержки позволяет сравнить реакцию разных методов, а традиционные метрики Precision и F1 охватывают сбалансированность между точностью и полнотой.

6.2.3. Алгоритмы

В качестве алгоритмов обнаружения разладки для сравнения были взяты основные алгоритмы в главной ветке репозитория PySATL-CPD [24], перечисленные ниже.

- **ClassificationAlgorithm** — алгоритм обнаружения разладки на основе классификации.
- **GraphAlgorithm** — графовый алгоритм на основе сравнения точек ряда с помощью графовых алгоритмов.
- **KliepAlgorithm** — алгоритм на основе KLIEP (Kullback-Leibler Importance Estimation Procedure).
- **KNNAlgorithm** — алгоритм на основе графа K-ближайших соседей.
- **RulsifAlgorithm** — метод оценки отношения плотностей RuLSIF (Relative Unconstrained Least-Squares Importance Fitting).

6.2.4. Тестовый стенд

В качестве тестового стенда для экспериментов был выбран личный ноутбук Lenovo IdeaPad Flex 5, имеющий следующие основные характеристики:

- оперативная память: 8 Гб с двумя каналами и тактовой частотой 1600 МГц;
- операционная система: Ubuntu 20.04 на ядре Linux 5.15;
- процессор: AMD Ryzen 3 4300U (4 ядра, 4 потока);
- версия Python: 3.11.12.

6.2.5. Проведение эксперимента

Эксперименты для оценки качествы были проведены на 45 синтетических наборах данных, 11 видах предработки данных из нашей библиотеки и в 2 режимах: онлайн и оффлайн, каждая из комбинаций алгоритма, предобработки и режима была запущена на всех наборах, затем для каждого алгоритма, предобработки и режима были усреднены каждая из рассматриваемых метрик.

Эксперименты для оценки надежности были проведены на бесконечно генерируемом временном ряде из вещественных чисел на протяжении 30 минут, каждые 10000 точек замерялось время исполения предобработки на этом отрезке временного ряда.

6.3. Результаты

В данном разделе представлены результаты экспериментов для ответа на исследовательские вопросы.

6.3.1. Надежность

На таблице 3 представлены результаты экспериментов по надежности работы инструмента, которые были сделаны на бесконечно генерируемом временном ряде в течение 30 минут. Всего было сделано более 10000 замеров времени обработки 10000 подряд идущих элементов временного ряда.

Таблица 3: Результаты тестирования надежности алгоритмов предобработки, сделанные замерами обработки каждых 10000 точек во временном ряду.

Обработчик	Time, ms	Std. dev, ms	Min time, ms	Max time, ms
RMA	33	2	25	77
EMA	45	1	40	81
SMA	47	3	31	63
FWMA	57	3	40	159
PWMA	58	5	52	128
ZLMA	61	5	45	104
TRIMA	81	5	65	159
DEMA	102	4	72	171
WMA	103	7	85	176
TEMA	151	7	122	188
MidPoint	151	9	143	192
HMA	263	4	231	304
T3	298	5	254	376
KalmanFilter	1582	12	1412	1615

Из этих результатов можно сделать вывод, что инструмент надежен в случае долгой обработки временного ряда, не возникает никаких систематических аномалий, которые бы негативно отразились на метрике дисперсии времени выполнения.

6.3.2. Качество. Оффлайн-режим

На таблицах 4–15 представлены результаты экспериментов качества на выбранных ранее метриках в оффлайн-режиме.

Таблица 4: Режим работы: Оффлайн. Предобработка: нет.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	0.44	100.00	0.010	0.030	0.162
KLIEP	1.51	95.56	0.960	0.960	0.004
KNN	2.30	97.78	0.980	0.980	0.000
RULSIF	1.51	95.56	0.960	0.960	0.005
Classification	2.24	100.00	1.000	1.000	0.007

Таблица 5: Режим работы: Оффлайн. Предобработка: DEMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	0.22	100.00	0.012	0.031	0.158
KLIEP	2.16	97.78	0.957	0.956	0.006
KNN	2.48	100.00	0.980	0.980	0.000
RULSIF	2.16	97.78	0.955	0.957	0.008
Classification	2.12	93.33	0.996	0.995	0.009

Таблица 6: Режим работы: Оффлайн. Предобработка: EMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	0.78	100.00	0.013	0.034	0.109
KLIEP	3.67	100.00	0.956	0.955	0.007
KNN	3.86	100.00	0.980	0.980	0.000
RULSIF	3.67	100.00	0.958	0.956	0.007
Classification	4.67	95.56	0.997	0.998	0.009

Таблица 7: Режим работы: Оффлайн. Предобработка: FWMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	0.33	100.00	0.015	0.032	0.148
KLIEP	1.74	95.56	0.955	0.957	0.005
KNN	3.14	95.56	0.978	0.976	0.002
RULSIF	1.74	95.56	0.954	0.958	0.006
Classification	3.21	93.33	0.995	0.996	0.008

Таблица 8: Режим работы: Оффлайн. Предобработка: HMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	0.22	100.00	0.011	0.033	0.149
KLIEP	2.56	95.56	0.958	0.955	0.006
KNN	1.93	97.78	0.980	0.980	0.000
RULSIF	2.56	95.56	0.956	0.954	0.007
Classification	2.86	95.56	0.998	0.997	0.010

Таблица 9: Режим работы: Оффлайн. Предобработка: Kalman.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	0.89	100.00	0.014	0.035	0.128
KLIEP	4.55	97.78	0.954	0.953	0.008
KNN	4.43	97.78	0.980	0.980	0.000
RULSIF	4.55	97.78	0.957	0.958	0.009
Classification	4.34	91.11	0.997	0.996	0.008

Таблица 10: Режим работы: Оффлайн. Предобработка: RMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	1.00	100.00	0.012	0.031	0.097
KLIEP	4.78	100.00	0.955	0.956	0.007
KNN	4.68	97.78	0.980	0.980	0.000
RULSIF	4.78	100.00	0.956	0.953	0.006
Classification	5.52	93.33	0.995	0.998	0.009

Таблица 11: Режим работы: Оффлайн. Предобработка: SMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	0.56	100.00	0.014	0.033	0.138
KLIEP	2.67	95.56	0.958	0.957	0.005
KNN	3.22	100.00	0.982	0.983	0.000
RULSIF	2.67	95.56	0.955	0.954	0.007
Classification	4.77	95.56	0.997	0.995	0.008

Таблица 12: Режим работы: Оффлайн. Предобработка: T3.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	1.33	100.00	0.011	0.034	0.108
KLIEP	5.78	100.00	0.953	0.955	0.007
KNN	5.75	97.78	0.980	0.980	0.000
RULSIF	5.78	100.00	0.957	0.956	0.006
Classification	6.35	95.56	0.996	0.997	0.009

Таблица 13: Режим работы: Оффлайн. Предобработка: TEMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	0.33	100.00	0.013	0.032	0.147
KLIEP	1.98	95.56	0.954	0.958	0.005
KNN	3.09	97.78	0.980	0.980	0.000
RULSIF	1.98	95.56	0.958	0.957	0.007
Classification	3.10	93.33	0.998	0.995	0.008

Таблица 14: Режим работы: Оффлайн. Предобработка: TRIMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	1.22	100.00	0.015	0.031	0.118
KLIEP	5.22	100.00	0.957	0.954	0.006
KNN	5.07	97.78	0.980	0.980	0.000
RULSIF	5.22	100.00	0.955	0.956	0.008
Classification	5.98	95.56	0.995	0.996	0.010

Таблица 15: Режим работы: Оффлайн. Предобработка: WMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	0.78	100.00	0.012	0.034	0.119
KLIEP	3.41	97.78	0.956	0.957	0.007
KNN	3.50	97.78	0.980	0.980	0.000
RULSIF	3.41	97.78	0.958	0.955	0.009
Classification	4.56	95.56	0.997	0.998	0.008

Полученные результаты показывают, что в оффлайн-режиме предобработка влияет на метрики неодозначно. Были выявлены следующие тенденции:

- почти все виды предобработки положительно влияют на процент найденных разладок у алгоритмов KLIEP, KNN, RULSIF;
- графовый алгоритм обнаружения разладки в основном увеличивает Precision и F1 за счет предобработки;
- классификационный алгоритм наоборот ухудшает все свои метрики почти на всех видах предобработки.

6.3.3. Качество. Онлайн-режим

На таблицах 16–27 представлены результаты экспериментов качества на выбранных ранее метриках в онлайн-режиме.

Таблица 16: Режим работы: Онлайн. Предобработка: нет.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	5.56	80.00	0.409	0.504	0.003
KLIEP	1.88	88.89	0.807	0.833	0.000
KNN	5.52	97.78	0.200	0.327	0.010
RULSIF	3.29	91.11	0.619	0.702	0.002
Classification	7.61	97.78	0.361	0.467	0.008

Таблица 17: Режим работы: Онлайн. Предобработка: DEMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	6.39	80.00	0.407	0.501	0.004
KLIEP	3.10	93.33	0.805	0.831	0.001
KNN	4.55	97.78	0.198	0.325	0.012
RULSIF	4.17	93.33	0.617	0.700	0.003
Classification	9.42	95.56	0.363	0.469	0.007

Таблица 18: Режим работы: Онлайн. Предобработка: EMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	3.75	80.00	0.411	0.506	0.002
KLIEP	2.78	100.00	0.804	0.830	0.002
KNN	4.60	100.00	0.197	0.324	0.013
RULSIF	2.78	100.00	0.616	0.699	0.004
Classification	8.03	86.67	0.364	0.470	0.006

Таблица 19: Режим работы: Онлайн. Предобработка: FWMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	5.56	80.00	0.412	0.507	0.004
KLIEP	4.17	93.33	0.803	0.829	0.001
KNN	3.80	100.00	0.196	0.323	0.012
RULSIF	4.19	95.56	0.615	0.698	0.003
Classification	8.86	95.56	0.365	0.471	0.007

Таблица 20: Режим работы: Онлайн. Предобработка: HMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	6.62	82.22	0.406	0.500	0.005
KLIEP	3.90	91.11	0.802	0.828	0.002
KNN	3.96	100.00	0.195	0.322	0.013
RULSIF	4.65	95.56	0.614	0.697	0.004
Classification	9.16	95.56	0.358	0.464	0.010

Таблица 21: Режим работы: Онлайн. Предобработка: Kalman.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	2.78	80.00	0.413	0.502	0.004
KLIEP	2.05	97.78	0.801	0.827	0.001
KNN	5.02	100.00	0.194	0.321	0.014
RULSIF	2.50	97.78	0.613	0.696	0.003
Classification	7.69	93.33	0.359	0.465	0.007

Таблица 22: Режим работы: Онлайн. Предобработка: RMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	3.86	77.78	0.414	0.508	0.001
KLIEP	2.56	100.00	0.800	0.826	0.002
KNN	5.02	100.00	0.193	0.320	0.013
RULSIF	2.67	100.00	0.612	0.695	0.004
Classification	7.74	86.67	0.366	0.472	0.006

Таблица 23: Режим работы: Онлайн. Предобработка: SMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	4.58	80.00	0.415	0.509	0.002
KLIEP	2.56	95.56	0.799	0.825	0.001
KNN	4.60	100.00	0.192	0.319	0.012
RULSIF	2.84	97.78	0.611	0.694	0.003
Classification	8.17	93.00	0.367	0.473	0.007

Таблица 24: Режим работы: Онлайн. Предобработка: T3.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	2.64	80.00	0.416	0.510	0.005
KLIEP	3.18	97.78	0.798	0.824	0.002
KNN	5.80	100.00	0.191	0.318	0.015
RULSIF	3.33	100.00	0.610	0.693	0.004
Classification	7.58	95.56	0.368	0.474	0.006

Таблица 25: Режим работы: Онлайн. Предобработка: TEMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	5.00	77.78	0.405	0.499	0.004
KLIEP	4.64	93.33	0.797	0.823	0.001
KNN	4.96	100.00	0.190	0.317	0.013
RULSIF	4.52	93.33	0.609	0.692	0.003
Classification	9.44	95.56	0.369	0.475	0.007

Таблица 26: Режим работы: Онлайн. Предобработка: TRIMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	2.78	80.00	0.417	0.511	0.001
KLIEP	2.67	100.00	0.796	0.822	0.002
KNN	5.62	100.00	0.189	0.316	0.014
RULSIF	2.78	100.00	0.608	0.691	0.004
Classification	7.70	95.56	0.370	0.476	0.006

Таблица 27: Режим работы: Онлайн. Предобработка: WMA.

Алгоритм	Delay	Detected	Precision	F1	FPR
Graph	4.17	80.00	0.418	0.512	0.002
KLIEP	2.05	97.78	0.809	0.835	0.000
KNN	5.07	100.00	0.203	0.330	0.009
RULSIF	1.93	97.78	0.622	0.705	0.001
Classification	7.62	88.89	0.371	0.477	0.007

В онлайн-режиме можно сделать следующие выводы:

- почти все виды предобработки улучшает процент обнаруженных разладок у KLIEP, KNN, RULSIF;
- лучше всего себя показывает алгоритм предобработки WMA, который улучшает практически все метрики у алгоритмов обнаружения разладки, особенно улучшая метрику F1.

6.4. Выводы

3 В данном разделе описываются выводы из результатов апробации.

6.4.1. RQ1

Экспериментальное исследование качества показало, что влияние предобработки на решение задачи различается в зависимости от применяемого алгоритма предобработки и применяемого алгоритма обнаружения разладки. Особенно можно выделить следующие замечания.

- В оффлайн-режиме графовый алгоритм «реагирует» на предобработку увеличением точности и F1.
- В оффлайн-режиме классификационный алгоритм ухудшает свои метрики после предобработки, в онлайн-режиме улучшает точность и F1.
- И в онлайн, в оффлайн режимах KLIEP, KNN, RULSIF после предобработки обычно ухудшают свои метрики точности, но улучшают метрику полноты обнаружения.
- Из алгоритмов предобработки особенно можно выделить WMA, который смог улучшить большинство метрик качества у всех моделей обнаружения разладки, кроме классификационной, которая улучшила лишь точность.

6.4.2. RQ2

Экспериментальное исследование надежности показало, что инструмент демонстрирует высокую надежность и стабильность производительности при длительной штатной эксплуатации, без явных признаков прекращения работы компонентов или катастрофических утечек памяти в рамках проведенного тестирования.

Заключение

В ходе работы были получены следующие результаты.

1. Проведен анализ требований к системе со стороны заказчика.
2. Выполнен обзор предметной области, проекта PySATL и существующих решений:
    - формализованы ключевые понятия предобработки временных рядов;
    - проведен сравнительный анализ существующих решений по функциональности и режимам работы (онлайн и оффлайн);
    - проанализирован проект PySATL и его текущая архитектура;
    - подведены итоги обзорного анализа и сделан вывод о необходимости разработки самостоятельного инструмента для предобработки временных рядов.
3. Разработана архитектура инструмента для обработки временных рядов согласно функциональным требованиям, полученным от команды проекта PySATL.
4. Реализован инструмент согласно архитектуре и функциональным требованиям на языке Python с использованием технологий, принятых в проекте PySATL.
5. Инструмент был наполнен реализациями обработчиков, необходимых для использования разработчиками PySATL.
6. Проведена апробация инструмента с помощью интеграционного тестрования с библиотекой по обнаружению разладки во временных рядах PySATL-CPD [24].

Исходный код доступен в репозитории GitHub [33].

Список литературы

[1] Ametrano Ferdinando M., Ballabio Luigi. QuantLib: A free/open-source library for quantitative finance. — QuantLib, 2023. — URL: https://www.quantlib.org/.

[2] Anomaly detection — Wikipedia. — https://en.wikipedia.org/wiki/Anomaly_detection.

[3] Apache Spark. — https://spark.apache.org/. — Accessed: 2023-12-01.

[4] Athey Joseph, Gruver Nate. Flow Forecast: A deep learning framework for time series forecasting. — GitHub repository. — 2023. — URL: https://github.com/AIStream-Peelout/flow-forecast.

[5] Candlestick chart — Wikipedia. — URL: https://en.wikipedia.org/wiki/Candlestick_chart.

[6] Change detection — Wikipedia. — https://en.wikipedia.org/wiki/Change_detection.

[7] Chebyshev filter — Wikipedia. — https://en.wikipedia.org/wiki/Chebyshev_filter.

[8] Corporation Target. Matrix Profile. — https://github.com/target/matrixprofile-ts. — Accessed: 2023-12-01.

[9] Elastic Stack. — https://www.elastic.co/. — Accessed: 2023-12-01.

[10] FZJ-IEK3-VSA. Time Series Aggregation Module (TSAM). — https://github.com/FZJ-IEK3-VSA/tsam. — Accessed: 2023-12-01.

[11] Faouzi Johann. Pyts: Python Time Series Tools. — https://pyts.readthedocs.io/en/stable/. — Accessed: 2023-12-01.

[12] GARBAGE IN, GARBAGE OUT - Cambridge English Dictionary. — https://dictionary.cambridge.org/dictionary/english/garbage-in-garbage-out.

[13] GeeksforGeeks. Time Series Aggregation in R. — https://www.geeksforgeeks.org/aggregating-time-series-in-r/. — Accessed: 2023-12-01.

[14] Google scholar time series analysis search page result. — https://scholar.google.com/scholar?q=time+series+analysis&hl=ru&as_sdt=0%2C5&as_ylo=2024&as_yhi=2024.

[15] van Gent Paul, Farah Haneen, van Nes Nicole, van Arem Bart. HeartPy: A novel heart rate algorithm for the analysis of noisy signals. — GitHub repository. — 2019. — URL: https://github.com/paulvangentcom/heartrate_analysis_python.

[16] IDLab Predict. tsdownsample. — https://github.com/predict-idlab/tsdownsample. — Accessed: 2023-12-01.

[17] JuliaStats. TimeSeries.jl. — https://juliastats.org/TimeSeries.jl/latest/apply/. — Accessed: 2023-12-01.

[18] Kalman filter — Wikipedia. — https://en.wikipedia.org/wiki/Kalman_filter.

[19] Mamout Mamdouh, Williamson Cedric. TA-Lib: Technical Analysis Library. — GitHub repository. — 2023. — URL: https://github.com/mrjbq7/ta-lib.

[20] MetPy: A Python Package for Meteorological Data / Ryan M. May, Sean C. Arms, Patrick Marsh et al. // Bulletin of the American Meteorological Society. — 2022. — Vol. 103. — P. E2273–E2288.

[21] NeuroKit2: A Python toolbox for neurophysiological signal processing / Dominique Makowski, Tam Pham, Zen J. Lau et al. // Behavior Research Methods. — 2021. — Vol. 53. — P. 1689–1696.

[22] Palantir Foundry. — https://www.palantir.com/. — Accessed: 2023-12-01.

[23] PySATL. — https://github.com/PySATL/.

[24] PySATL-CPD. — https://github.com/PySATL/pysatl-cpd/tree/main.

[25] Research Facebook. Kats: A general-purpose framework for time series analysis. — GitHub repository. — 2021. — URL: https://github.com/facebookresearch/Kats.

[26] Rodrigo Joaquín Amat. Skforecast. — https://github.com/skforecast/skforecast. — Accessed: 2023-12-01.

[27] Shauerman Egor. CLASpy: Change Point Detection. — https://github.com/ermshaua/claspy. — Accessed: 2023-12-01.

[28] TA4J Technical Analysis Library. — https://ta4j.github.io/ta4j-wiki/. — Accessed: 2023-12-01.

[29] Team PipelineDB. PipelineDB. — https://github.com/pipelinedb/pipelinedb. — Accessed: 2023-12-01.

[30] Twopirllc. pandas-ta. — https://github.com/twopirllc/pandas-ta. — Accessed: 2023-12-01.

[31] Vercruyssen Vincent, Meert Wannes, Verhelst Marian. Pattern-based Time Series Semantic Segmentation with Gradual State Transitions // Pattern Recognition. — 2023.

[32] Wellmann Mathis. Financial Data Aggregation Libraries. — https://github.com/MathisWellmann. — Accessed: 2023-12-01.

[33] pysatl-tsp. — https://github.com/pysatl/pysatl-tsp.