Программное обеспечение «Kolmogorov.ai Predicate»

Руководство по эксплуатации

ООО «Дата Сапиенс» | kolmogorov.ai

ПО «Kolmogorov.ai Predicate» | version 2.1.0

Содержание

1. О Predicate 3
1.1 Назначение и функционал 3
1.1.1 Роль Predicate в промышленном применении модели 3
1.1.2 Функционал 3
1.2 Глоссарий 4
1.3 Обзор основного меню 5
1.3.1 Основной набор функций 5
1.3.2 Дополнительные функции меню 6
2. Объекты 8
2.1 Источники данных 8
2.1.1 Поддерживаемые источники данных 8
2.1.2 Подключение нового источника данных 9
2.2 Данные 12
2.2.1 Регистрация нового датасета 12
2.3 Метрики 14
2.3.1 Реестр тестов и метрик 14
2.3.2 Регистрация новой метрики 91
2.3.3 Правила написания метрик 93
2.3.4 Параметры метрик Predicate 99
2.3.5 Примеры кода метрик 100
2.4 Преобразования 112
2.4.1 Регистрация нового преобразования 112
2.4.2 Правила написания преобразований 114
2.5 Проекты 116
2.5.1 Создание нового проекта 116
2.5.2 Просмотр результатов мониторинга 127
2.6 Отчёты 128
2.6.1 Создание отчёта 128

1. О Predicate

1.1 Назначение и функционал

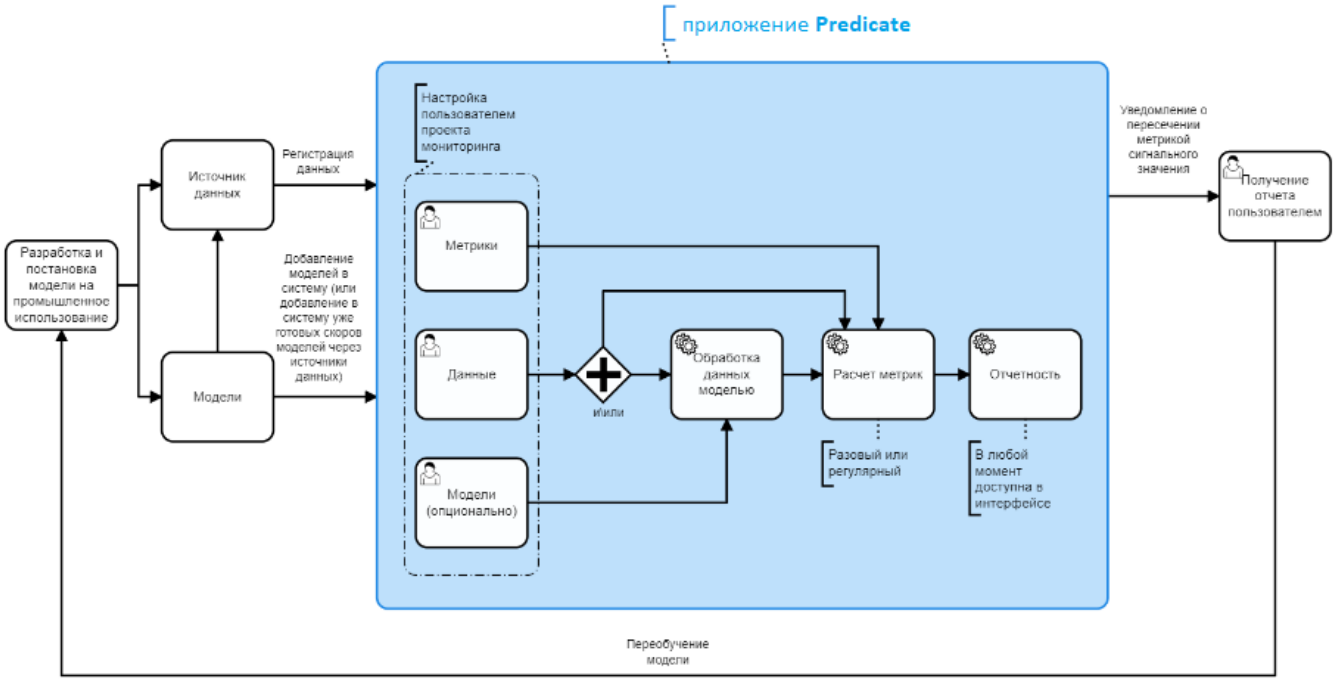
Predicate [ˈprɛdɪkɪt] - Инструмент класса *Metrics Store*.

Позволяет организовать библиотеку произвольных метрик и их визуализаций, упрощает задачу регламентного построения рабочих панелей мониторинга бизнес-решений и обработки инцидентов качества.

Приложение Kolmogorov.ai **Predicate** может применяться как инструмент класса *Model Quality Management* при автоматизации процессов тестирования, мониторинга и валидации аналитических моделей и наборов данных в системах принятия решений различных индустрий:

- финансовый сектор;
- телеком, ритейл;
- производство.

1.1.1 Роль Predicate в промышленном применении модели



1.1.2 Функционал

В приложении Kolmogorov.ai **Predicate** предусмотрены следующие **пользовательские сценарии**:

- постановка модели на мониторинг с автоматическим созданием дашборда;
- регистрация нового объекта данных;
- загрузка новой метрики или теста;
- загрузка модели или преобразования данных;
- создание шаблона проекта мониторинга;
- создание отчета о валидации.

1.2 Глоссарий

Источник данных - подключение к некоторому хранилищу данных (файлов / таблиц БД).

Данные - датасет с сохраненными настроенными метаданными (источник подключения, описание, разметка и т.д.). Примеры объектов данных: один файл csv, одна таблица SQL БД.

Разметка данных - описание свойств столбцов зарегистрированного в системе объекта данных.

Метрика - исполняемый над данными тест (скрипт), выводящий результат в виде графика и (или) скалярного значения.

Модель - предиктивная аналитическая модель.

Преобразование - исполняемый над данными скрипт, выводящий результат в виде датасета с числом столбцов, большим или равным числу столбцов в исходном датасете. Применение модели к датасету, содержащему набор признаков, задается в системе как преобразование, добавляющее к исходному датасету столбец с предсказаниями.

Проект - проект мониторинга. Состоит из набора данных, метрик и (опционально) моделей с настроенным пользователем режимом исполнения.

Регламент - расписание запусков проекта мониторинга.

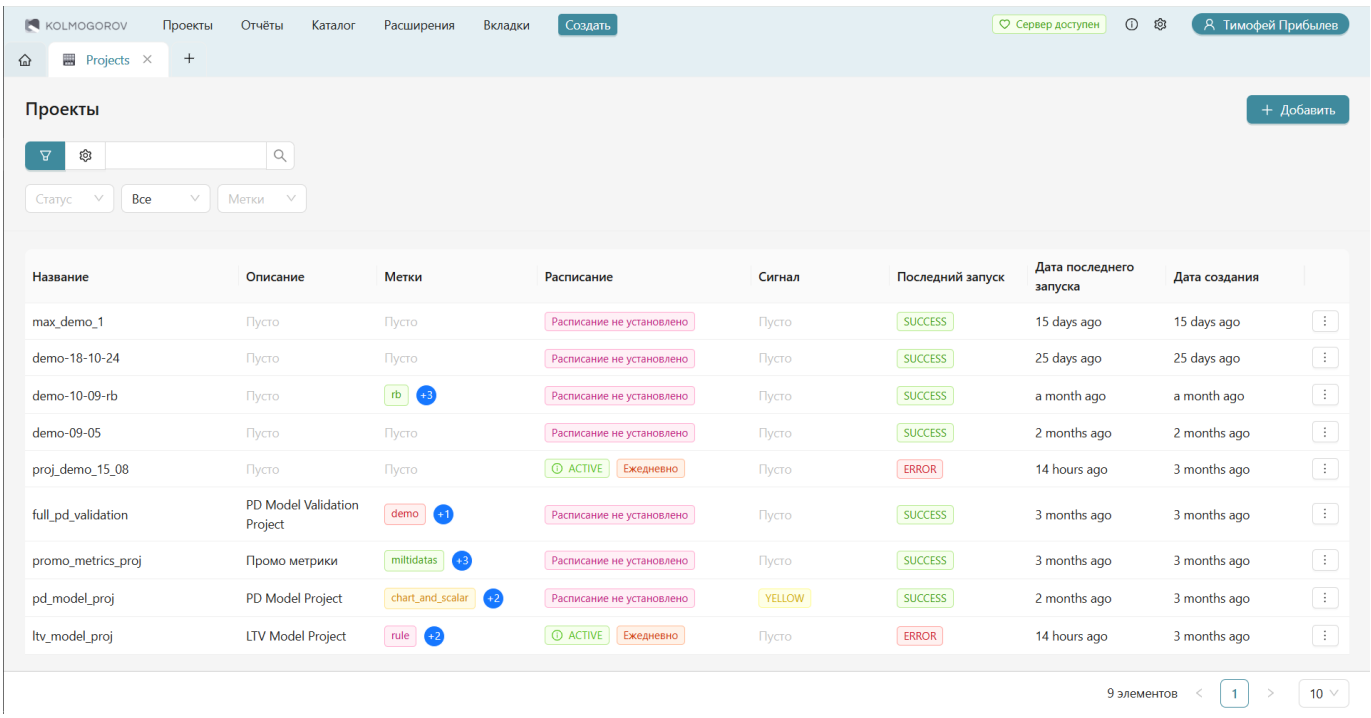
Дашборд - раздел проекта с результатами расчета метрик.

Пользовательский дашборд (отчет) - совокупность текстовых панелей и графиков, созданная пользователем на основе результатов расчета проектов мониторинга. Один пользовательский отчет может включать результаты нескольких проектов мониторинга.

Шаблон проекта - преднастроенный сохраненный набор метрик, их параметров и (опционально) настроек регламента.

1.3 Обзор основного меню

Основное меню приложения расположено в верхней части экрана:



Описание разделов меню и функций, доступных из этих разделов, приведено ниже.

Ниже основного меню расположена панель вкладок, в которой доступны все ранее открытые страницы каталогов и объектов.

1.3.1 Основной набор функций

Проекты

Каталог имеющихся в системе (созданных ранее) проектов мониторинга качества моделей/данных.

Для каждого проекта доступны:

- основная информация (название, описание, статус, дата создания и др.);
- список входных параметров (используемые данные и метрики);
- расписание запусков;
- результаты выполнения проекта (автоматически формируемый дашборд и логи).

Отчеты

Каталог пользовательских отчетов, которые собираются из доступных в системе результатов проектов мониторинга (графиков, скалярных значений метрик и светофоров). Также отчеты позволяют включать блоки с текстовой информацией, пользовательские изображения агрегирующие таблицы по проектам мониторинга.

Доступна выгрузка отчета в файл pdf.

Каталог

Кнопка "Каталог" вызывает выпадающий список со следующими кнопками:

- Данные - Каталог датасетов, доступных для проведения расчетов в проектах мониторинга.
- Шаблоны - Каталог шаблонов проектов мониторинга.
- Метрики - Каталог метрик (тестов), доступных для расчета в проектах мониторинга.
- Преобразования - Каталог преобразований, доступных для использования в проектах мониторинга.
- Источники данных - Каталог подключений к системам хранения данных, доступных в системе.

Расширения

Кнопка "Расширения" вызывает выпадающий список со следующими кнопками:

- Celery monitoring - административная панель для просмотра технических задач за период времени или отдельный запуск проекта.
- Celery schedule - административная панель для просмотра технического состояния проектов на расписании.
- Сравнение проектов - страница для сравнения метрик из запусков проектов.
- Календарь - страница с календарем запуска регулярных проектов.

Вкладки

Кнопка "Вкладки" вызывает выпадающий список со следующими кнопками:

- Добавить в избранное / Удалить из избранного - кнопка для добавления открытой вкладки в избранное или ее удаление из избранного.
- Закрыть все вкладки - кнопка для закрытия всех вкладок.
- Избранное - выпадающий список с избранными вкладками.

Создать

Кнопка "Создать" вызывает выпадающий список со следующими кнопками:

- Проект - Страница создания проекта мониторинга качества модели/данных, либо создания шаблона такого проекта.
- Отчет - Страница создания пользовательского отчета
- Данные - Страница создания нового датасета.
- Утилиты - выпадающий список с дополнительными кнопками:
- Метрика - Страница создания новой метрики.
- Преобразование - Страница создания нового преобразования.
- Источник данных - Страница подключения нового источника данных.

Подразумевает выбор пользователем данных для расчета, необходимых метрик и периодичности мониторинга, а также (опционально) настройку светофора проекта.

1.3.2 Дополнительные функции меню

В меню также содержится информация о доступности сервера Predicate, о версии инструмента, и доступны дополнительные настройки.

Настройки

Настройки представляют из себя набор переключателей для изменения параметров отображения интерфейса.

Доступны следующие настройки:

- Язык: Английский или Русский - переключает язык интерфейса.
- Тема: Светлая или Темная - переключает цветовую тему интерфейса.

Выход из системы

Кнопка для выхода из системы данного пользователя.

2. Объекты

2.1 Источники данных

2.1.1 Поддерживаемые источники данных

Источник данных - подключение к существующему хранилищу данных, позволяющее получить доступ к находящимся в хранилище таблицам данных для дальнейшего формирования датасетов, которые будут использоваться при создании проектов мониторинга.

Загрузка данных для проектов мониторинга происходит из реляционных источников, зарегистрированных в системе.

Продукт поддерживает следующие типы подключения:

- хранилище **s3** (хранение файлов типа .csv)
- базы данных **PostgreSQL**, **Oracle**, **Hive**
- feature store **Kolmogorov.Axiom**

Для использования в проекте мониторинга, датасет из источника данных должен быть зарегистрирован в системе согласно алгоритму регистрации датасета.

Примечание

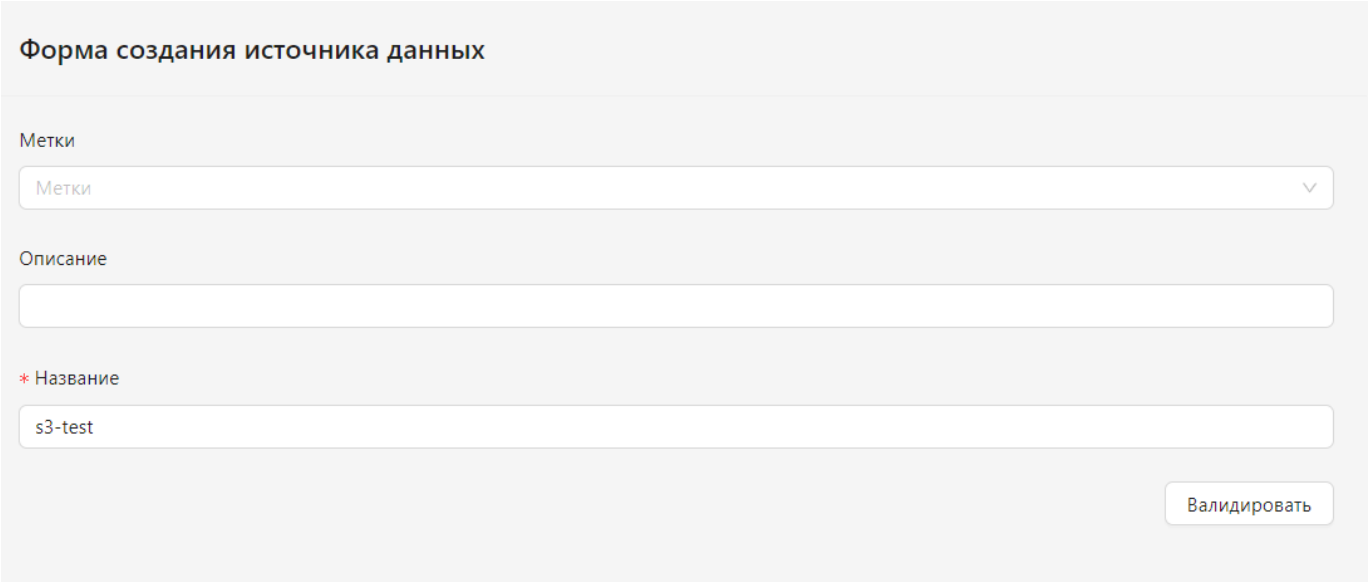
При создании проекта мониторинга на данных из динамично пополняемой таблицы PostgreSQL возможно настроить **границы по времени** для среза данных, используемого при расчете метрики.

Примечание

Загрузка данных с рабочих машин пользователей возможна в хранилища s3 в формате .csv.

2.1.2 Подключение нового источника данных

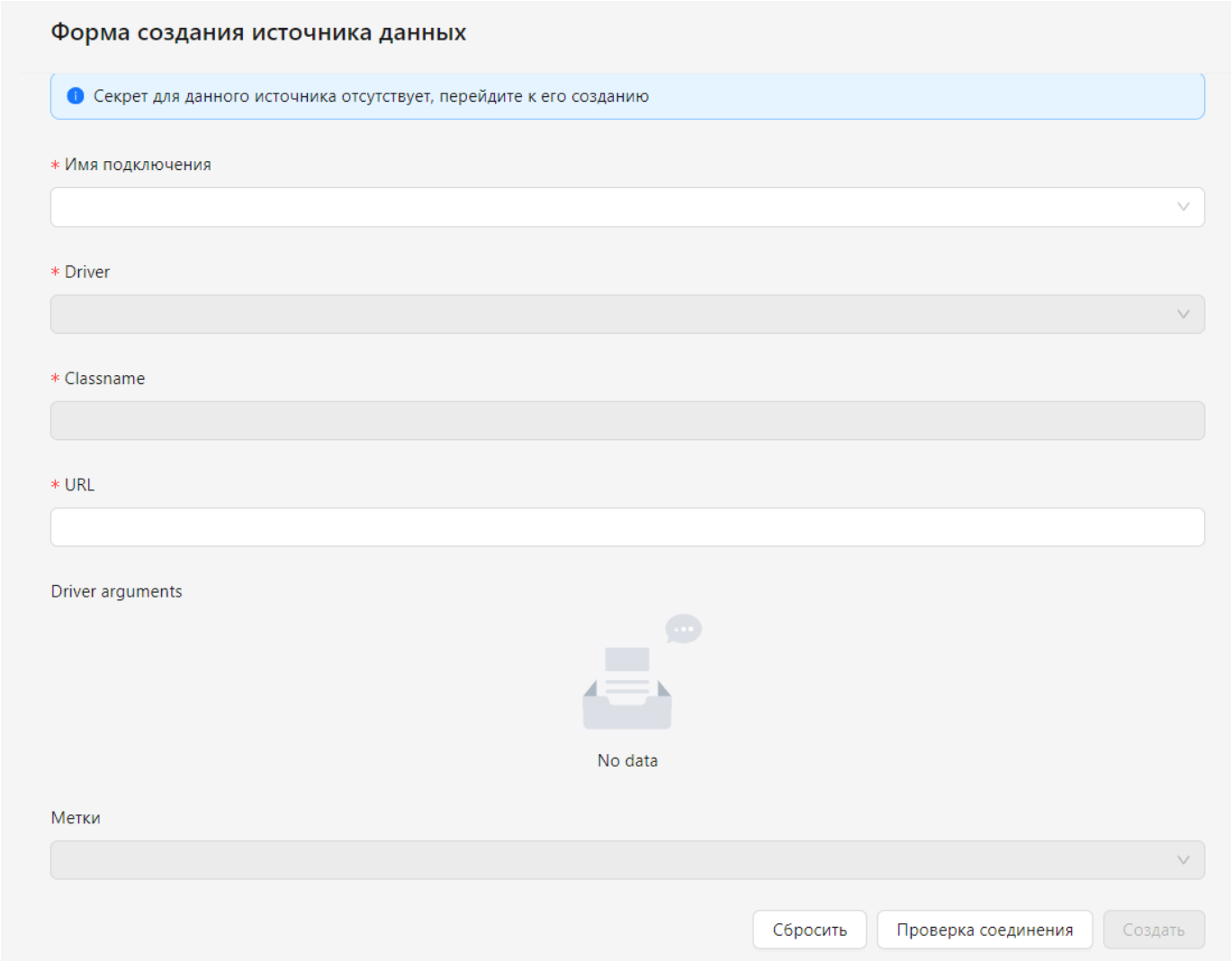
Создание нового источника данных доступна через интерфейс: *Панель управления* > *Каталог* > *Источники данных* > *Добавить* или *Панель управления* > *Создать* > *Утилиты* > *Источник данных*



Параметры регистрации:

- Название (обязательное поле) - название источника данных в системе. Должно содержать только латинские символы, цифры и дефис. По этому названию будет создан секрет в namespace, где развернуть Predicate, поэтому данное имя не должно соответствовать существующим секретам.
- Описание (необязательное поле) - описание источника данных.
- Метки (необязательное поле) - список меток источника данных.

После заполнения названия необходимо нажать кнопку "Валидировать". Если секрет с указанным названием уже существует - появится ошибка. Если имя доступно, то появляется дополнительные поля для параметризации секрета.



Параметры секрета:

- Имя подключения (обязательное поле) - название источника данных. Доступны следующие опции:
- s3-duckdb - подключение к источнику S3;
- postgresql - подключение к БД PostgreSQL;
- oracle - подключение к БД Oracle;
- hive - подключение к БД Hive.
- Driver (обязательное поле) - драйвер подключения к источнику, заполняется автоматически после выбора имени подключения.
- Classname (обязательное поле) - класс подключения к источнику, заполняется автоматически после выбора имени подключения.
- URL (обязательное поле) - путь подключения к источнику. Для s3-duckdb заполняется автоматически, для остальных подключений появляется шаблон заполнения данных.
- Driver Arguments - перечень значений параметров драйвера. Конкретный список появляется после выбора имени подключения.
- Метки - список автоматических метрик, которые назначаются в зависимости от имени подключения.

В зависимости от имени подключения Driver Arguments могут содержать следующие поля:

Имя подключения	Driver Arguments
s3-duckdb	1. Путь до S3 (s3_url) 2. Ключ подключения (s3_access_key) 3. Секрет подключения (s3_secret_access_key) 4. Название бакета (s3_bucket)
postgresql	1. Логин (user) 2. Пароль (password)
oracle	1. Логин (user) 2. Пароль (password)
hive	1. Логин (user) 2. Пароль (password)

После заполнения всех данных необходимо нажать на кнопку "Проверка соединения" для верификации введенных данных. После успешного соединения необходимо нажать кнопку "Создать".

Результат регистрации: параметры источника данных становятся доступны через *Панель управления > Каталог > Источники данных*. В списке источников отображается статус успешности подключения.

2.2 Данные

2.2.1 Регистрация нового датасета

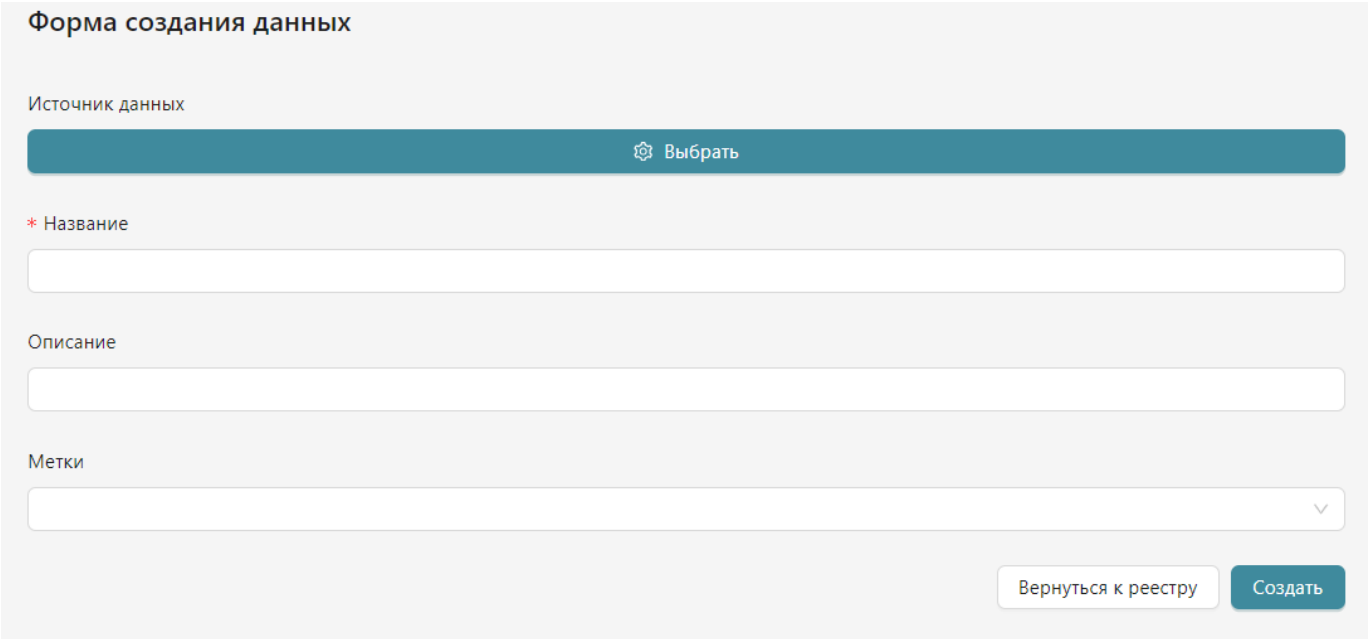
В данном разделе описан процесс регистрации датасета из источника, подключенного к системе. Подробнее про источники данных см. раздел "Источники данных".

Важно

На данный момент существует **ограничение** на объем датасета - **1 Гб**. Система не тестировалась на работу с файлами большего размера. Рекомендуемый размер файла - до **100 Мб**.

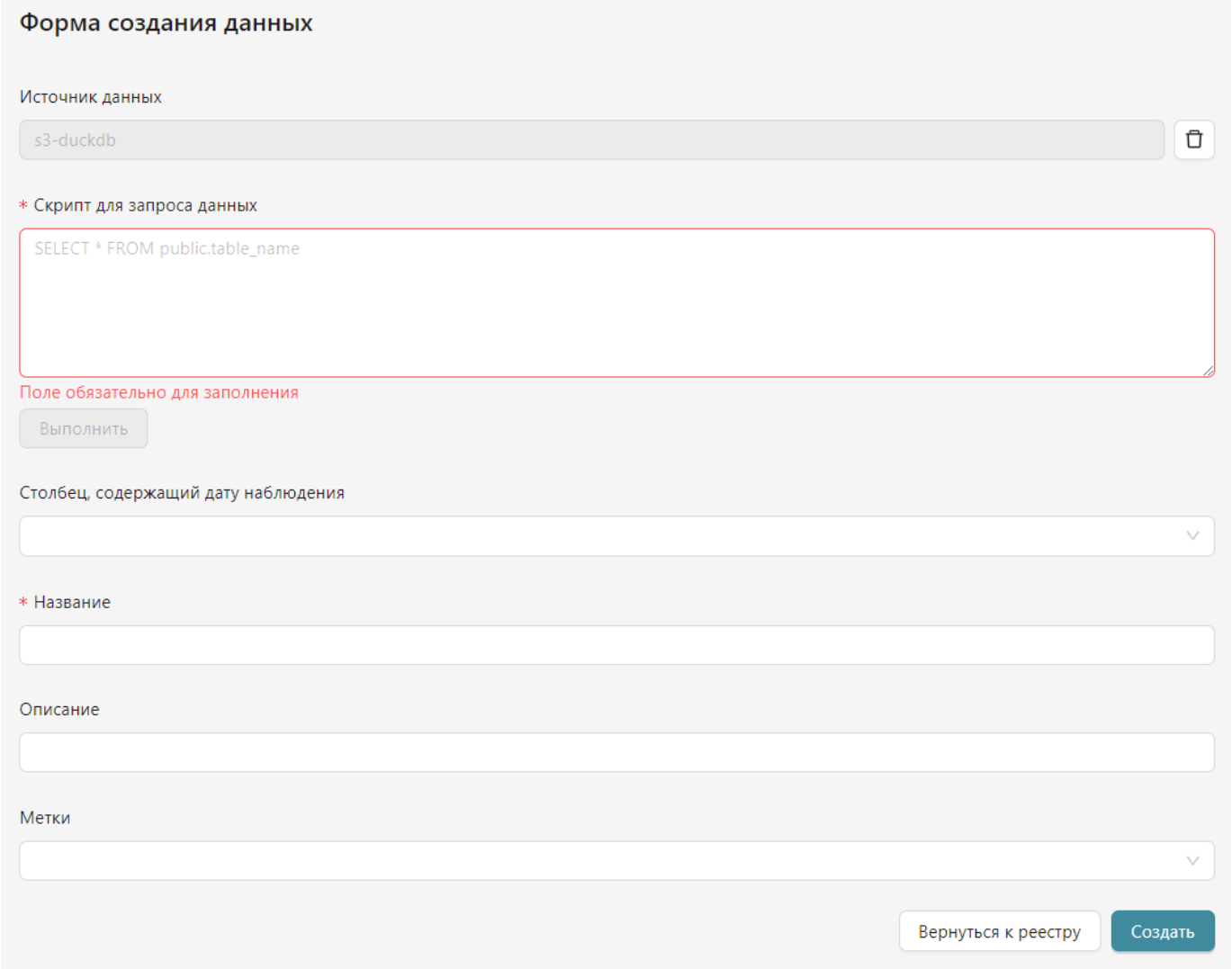
Перейдите по пути *Каталог > Данные* в основном меню приложения. В открывшемся окне каталога данных нажмите кнопку "Добавить".

Появится форма регистрации нового датасета:



В первую очередь необходимо выбрать источник данных. При нажатии на кнопку "Выбрать" появится каталог, из которого можно будет выбрать нужный источник.

После выбора источника форма создания датасета обновится:



Параметры регистрации:

- Скрипт для запроса данных (**обязательное поле**) - скрипт на том диалекте SQL, который воспринимается источником данных. В случае с файловым сервисом S3 вместо названия таблицы необходимо указать следующую функцию: `read_csv_auto("s3:///путь до файла в хранилище из корня>")` . Например, `select * from read_csv_auto("s3://klmg-bucket/df.csv")`
- Столбец, содержащий дату наблюдения (**необязательное поле**, заполняется строго после составления скрипта и нажатия кнопки "Выполнить") - название столбца, который содержит отчетную дату для данных. Необходим в том случае, если датасет регистрируется для использования в проекте мониторинга с плавающим окном мониторинга.
- Название (**обязательное поле**) - название датасета.
- Описание (**необязательное поле**) - описание датасета.
- Метки (**необязательное поле**) - метки (теги) датасета.
- Файл (**необязательное поле**) - поле для загрузки файла на S3. **Доступно только для источников данных S3**.

После заполнения всех данных необходимо нажать кнопку "Создать".

В дальнейшем к просмотру информации об этом датасете можно вернуться по схеме: *Панель управления > Каталог > Данные > [Выбор строки с названием датасета] > двойной клик*

В каталоге данных отображается статус успешности регистрации датасета.

2.3 Метрики

2.3.1 Реестр тестов и метрик

Core package (базовый функционал). Анализ данных

ТОЧЕЧНЫЕ ОЦЕНКИ

cd_1_1_Mean

- **Техническое название**: cd_1_1_Mean
- **Описание**: Mean. Среднее значение для выбранного столбца
- **Теги**: core, data, scalar
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Отношение суммы всех значений в столбце к числу значений.
Только для столбцов с числовыми данными.
Если в столбце есть пропуски, они исключаются из рассмотрения.

Входные параметры

df: датасет с данными для исследования (dataframe)
field_column: название столбца для расчета (column)
higher_is_better: Большее значение метрики - лучше (bool)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: не применимо, т.к. результат - скалярное выражение (число).
Output (long): Отсутствует
Output (short): Число:среднее значение
Output example (picture): не применимо, т.к. результат - скалярное выражение (число).

cd_1_2_Median

- **Техническое название**: cd_1_2_Median
- **Описание**: Median. Медиана для выбранного столбца
- **Теги**: core, data, scalar
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Число, которое делит упорядоченный набор значений выбранного столбца на две равные части.
Только для столбцов с числовыми данными.
Если в столбце есть пропуски, они исключаются из рассмотрения.

Входные параметры

df: датасет с данными для исследования (dataframe)
field_column: название столбца для расчета (column)
higher_is_better: Большее значение метрики - лучше (bool)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: не применимо.
Output (long): Отсутствует
Output (short): Число: значение медианы
Output example (picture): не применимо.

cd_1_3_Mode

- **Техническое название**: cd_1_3_Mode
- **Описание**: Mode. Мода и N популярных значений столбца
- **Теги**: core, data
- **requirements**: `typing`, `pandas`, , `plotly.graph_objects`
- **Примечания**: -

Логика исполнения

Наиболее часто встречающееся значение в выбранном столбце.
Для столбцов с **произвольным** типом данных.
Если в столбце есть пропуски, они исключаются из рассмотрения.

Входные параметры

df: датасет с данными для исследования (dataframe)
field_column: название столбца для расчета (column)
top_col_number: Число отображаемых популярных значений столбца

Результаты

Движок отрисовки графика: plotly.js

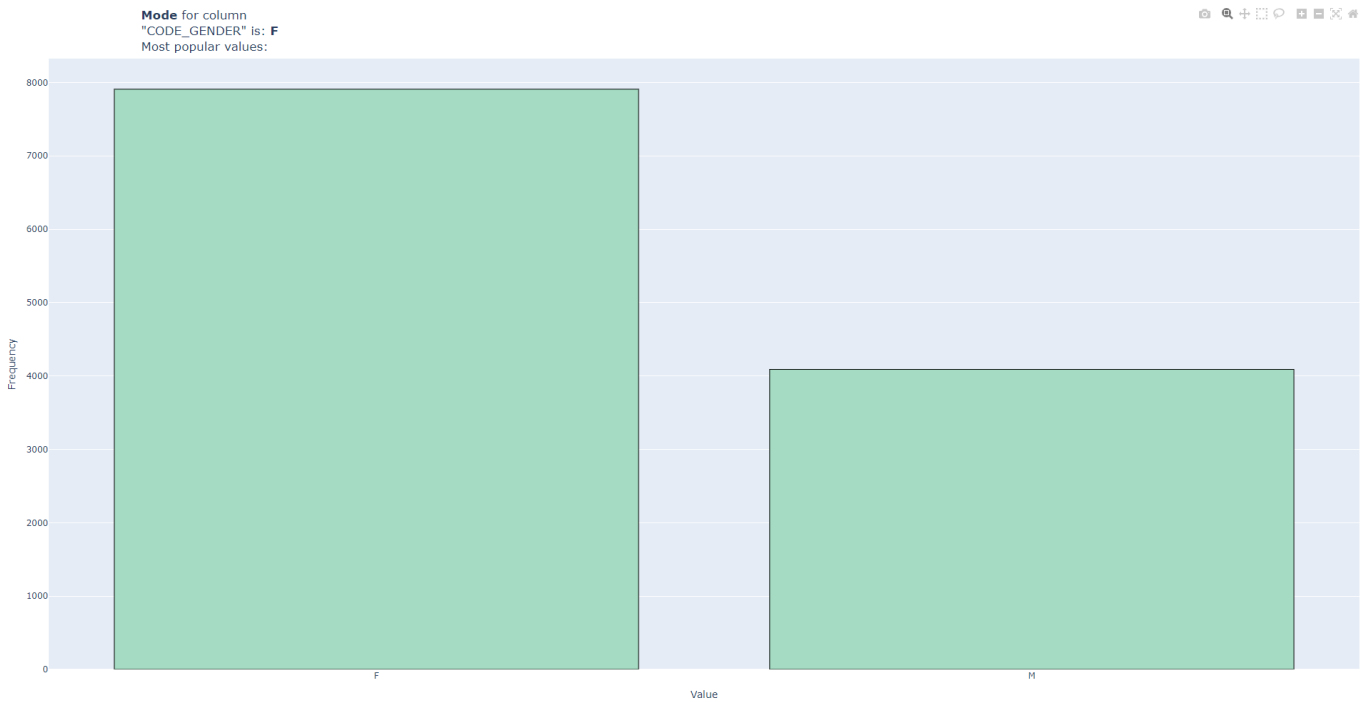
Output (long):

Barchart
xaxis: значения (N наиболее часто встречающихся значений столбца)
yaxis: частота встречаемости значения

Output (short):

Отсутствует

Output example (picture):



cd_1_4_Min

- **Техническое название**: cd_1_4_Min
- **Описание**: Min. Минимальное значение в выбранном столбце
- **Теги**: core, data, scalar
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Минимальное значение в выбранном столбце.
Только для столбцов с числовыми данными.
Если в столбце есть пропуски, они исключаются из рассмотрения.

Входные параметры

df: датасет с данными для исследования (dataframe)
field_column: название столбца для расчета (column)
higher_is_better: Большее значение метрики - лучше (bool)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Число:
минимальное значение в столбце

Output example (picture): не применимо.

cd_1_5_Max

- **Техническое название**: cd_1_5_Max
- **Описание**: Max. Максимальное значение в выбранном столбце
- **Теги**: core, data, scalar
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Максимальное значение в выбранном столбце.
Только для столбцов с числовыми данными.
Если в столбце есть пропуски, они исключаются из рассмотрения.

Входные параметры

df: датасет с данными для исследования (dataframe)
field_column: название столбца для расчета (column)
higher_is_better: Большее значение метрики - лучше (bool)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Число:
максимальное значение в столбце

Output example (picture): не применимо.

cd_1_6_Weighted_Prob

- **Техническое название**: cd_1_6_Weighted_Prob
- **Описание**: Weighted Probability. Взвешенная вероятность по предсказаниям модели
- **Теги**: core, data, scalar
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Для каждой записи (строки) вычисляем score * weight, суммируем полученные числа.
На выходе одно число - значение WP.

Входные параметры

df: датасет с данными для исследования (dataframe)
predict_column: название столбца со скором модели (column)
weight_column: название столбца с весами (column)
higher_is_better: Большее значение метрики - лучше (bool)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Число:
значение WP

Output example (picture): не применимо.

cd_1_7_Null_count

- **Техническое название**: cd_1_7_Null_count
- **Описание**: Null count. Количество Null значений в выбранном столбце
- **Теги**: core, data, scalar
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Количество Null значений в выбранном столбце

Входные параметры

df: датасет с данными для исследования (dataframe)
field_column: название столбца для расчета (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Число Null-значений в выбранном столбце

Output example (picture): не применимо.

БАЗОВЫЕ ОЦЕНКИ ПО ДАТАСЕТУ

cd_2_1_Df_Stats

- **Техническое название**: cd_2_1_Df_Stats
- **Описание**: Df Statistics Table. Базовые статистики по датафрейму

  (для df с небольшим числом столбцов)
- **Теги**: core, data
- **requirements**: `pandas`
- **Примечания**: -

Логика исполнения

Выводятся:
- число строк и число столбцов df
- число пропусков в каждом из столбцов
- для каждого из столбцов с числовыми данными: число записей, среднее значение, стандартное отклонение, min, max, перцентили: 25, 50 и 75.

Входные параметры

df: Объект данных для расчета (dataframe)

Результаты

Движок отрисовки графика: plotly.js

Output (long):

Таблица с базовыми статистиками по столбцам

Output (short):

Число строк в датафрейме
Число столбцов в датафрейме

Output example (picture):

col_name	count	mean	std	min	25%	50%	75%	max
client_id	11628	5250.7752	3158.7814	0	2441.75	5348.5	8255.25	9999
y_pred	11628	0.4739	0.4993	0	0	0	1	1
y_fact	11628	0.4878	0.4999	0	0	0	1	1
total_sum	11628	448.4191	501.8431	2.1204	108.2134	220.0095	640.0333	3537.6186
total_sum_	11628	437.2705	566.3495	0.916	92.4079	202.5112	562.1943	5989.9508
total_cnt	11628	0.0187	0.0069	0.0004	0.0143	0.0183	0.0228	0.0576

col_name	null_count
client_id	0
y_pred	0
y_fact	0
total_sum	0
total_sum_pred	0
total_cnt	0
report_dttm	0
date	0

Число строк:
11.63k

Число столбцов:
8

cd_2_2_Df_Stats_features

- **Техническое название**: cd_2_2_Df_Stats_features
- **Описание**: Statistics Table for Selected Columns. Статистики для выбранных столбцов
- **Теги**: core, data
- **requirements**: `pandas`
- **Примечания**: -

Логика исполнения

Выводятся:
count, mean, std, min, max, перцентили: 25, 50 и 75 для выбранных столбцов

Входные параметры

df: Объект данных для расчета (dataframe)
field_columns: Список названий столбцов для исследования (multi-column)

Результаты

Движок отрисовки графика: plotly.js

Output (long):

Таблица с базовыми статистиками по столбцам

Output (short):

Отсутствует

Output example (picture):

col_name	count	mean	std	min	25%	50%	75%	max
EVENT_PROBABILITY	11999	0.0794	0.0689	0.0072	0.0353	0.0582	0.0983	0.7166
TARGET	11999	0.0769	0.2665	0	0	0	0	1

cd_2_3_Density_Distr

- **Техническое название**: cd_2_3_Density_Distr
- **Описание**: Density Distribution. Плотность распределения для всех полей датасета
- **Теги**: core, data
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Цикл по всем столбцам df:
- если данные в столбце **не числовые**, пропускаем его;
- если данные **категориальные** (меньше или равно **categorial_threshold** различных значений), строится гистограмма;
- если данные **непрерывные**, строится линейный график плотности распределения.

Полученные гистограммы и графики выводятся в общее поле или на отдельные поля, в зависимости от значения **split_charts**. При выводе на общее поле нажатием на легенду можно **активировать нужный элемент**. Масштаб автоматически подстраивается под значения метрики.

Входные параметры

df: Объект данных для расчета (dataframe)
categorial_threshold: граница числа уникальных объектов в категориальной колонке датасета (int)
split_charts: флаг разделения графиков для разных колонок по разным карточкам (bool)

Результаты

Движок отрисовки графика: plotly.js

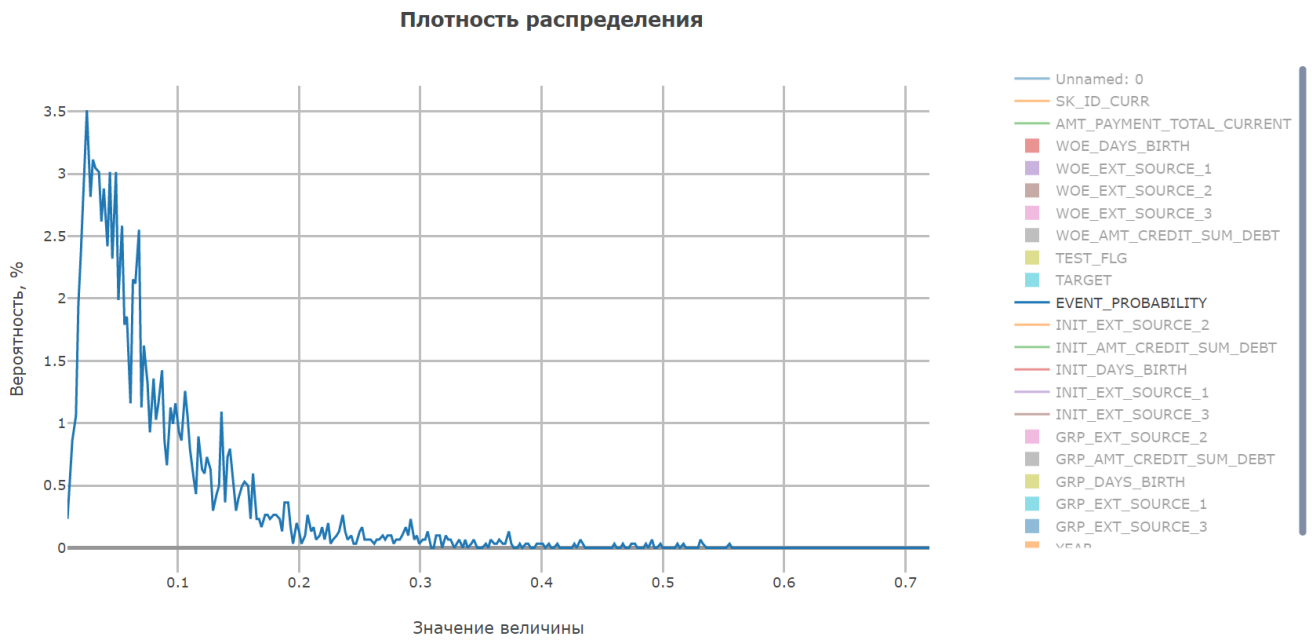
Output (long):

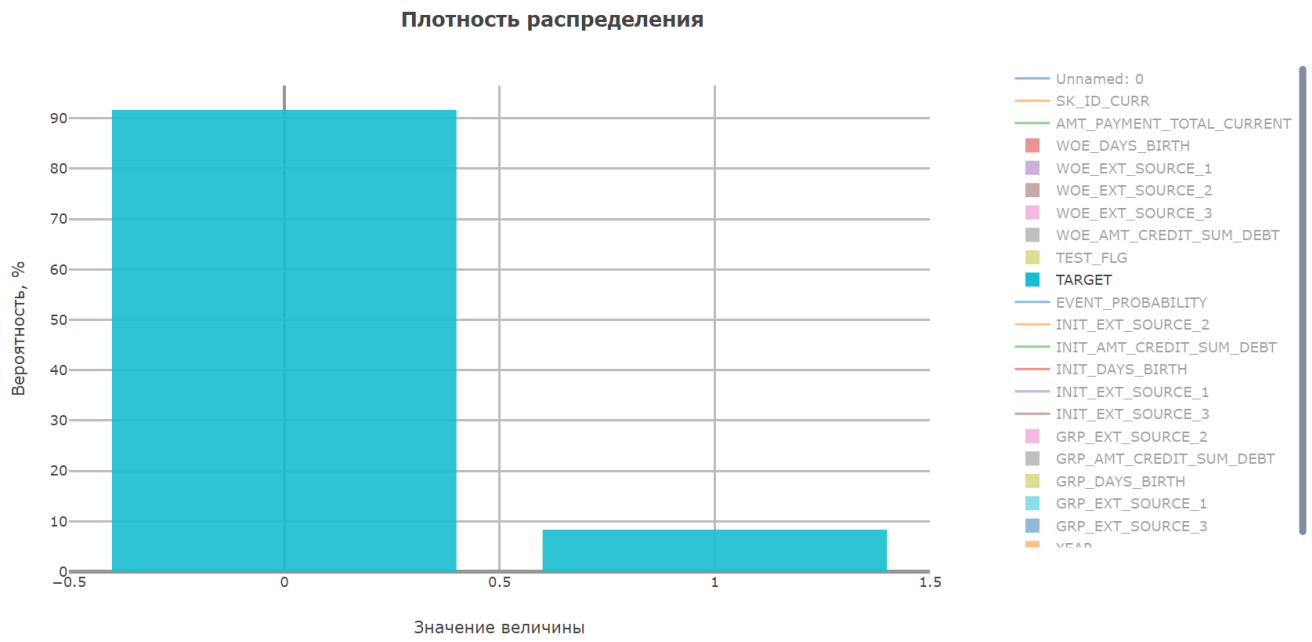
Массив графиков

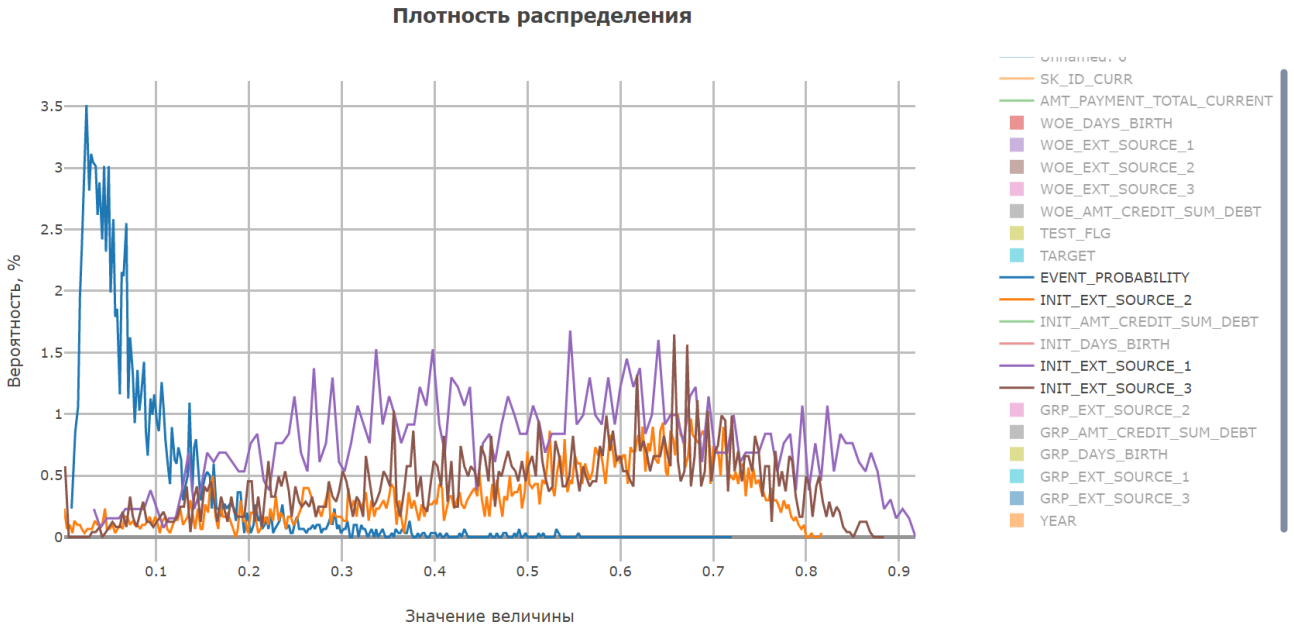
Output (short):

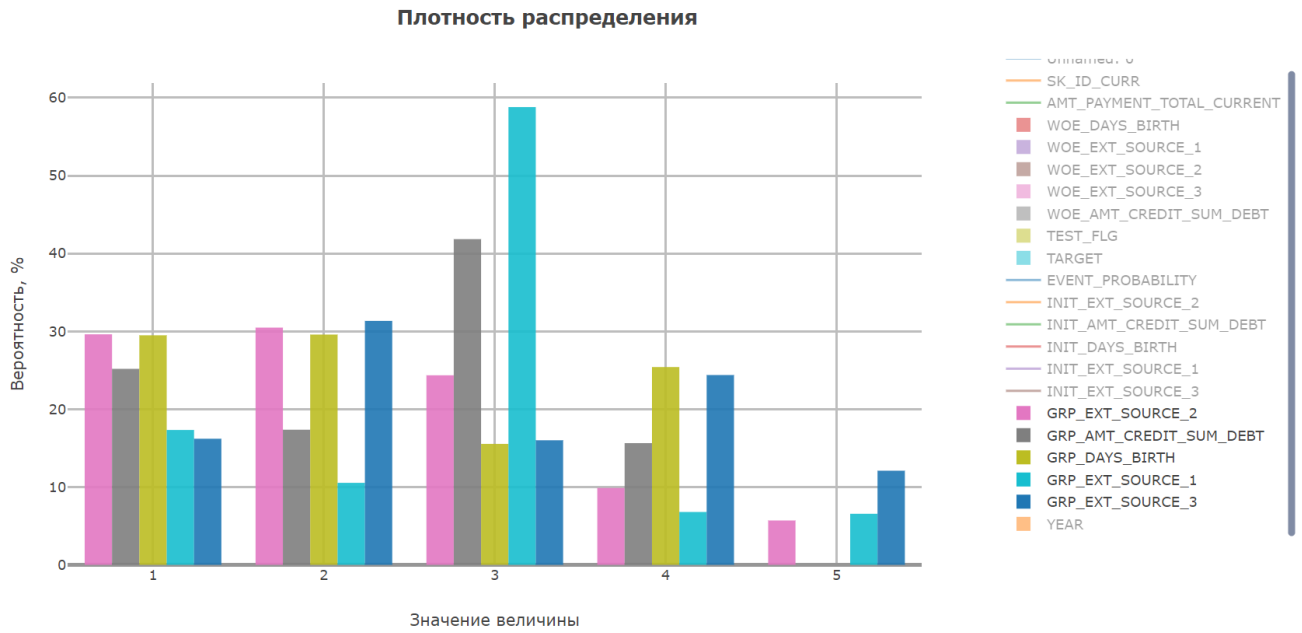
Отсутствует

Output example (picture):









cd_2_4_Density_Distr_features

- **Техническое название**: cd_2_4_Density_Distr_features
- **Описание**: Density Distribution for Selected Columns. Плотность распределения для выбранных полей
- **Теги**: core, data
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Цикл по выбранным столбцам df:
- если данные в столбце **не числовые**, пропускаем его;
- если данные **категориальные** (меньше или равно **categorial_threshold** различных значений), строится гистограмма;

- если данные **непрерывные**, строится линейный график плотности распределения.

Полученные гистограммы и графики выводятся в общее поле или на отдельные поля, в зависимости от значения **split_charts**. При выводе на общее поле нажатием на легенду можно **активировать нужный элемент**. Масштаб автоматически подстраивается под значения метрики.

Входные параметры

df: Объект данных для расчета (dataframe)
categorial_threshold: граница числа уникальных объектов в категориальной колонке датасета (int)
field_columns: Список названий столбцов для исследования (multi-column)
split_charts: флаг разделения графиков для разных колонок по разным карточкам (bool)

Результаты

Движок отрисовки графика: plotly.js

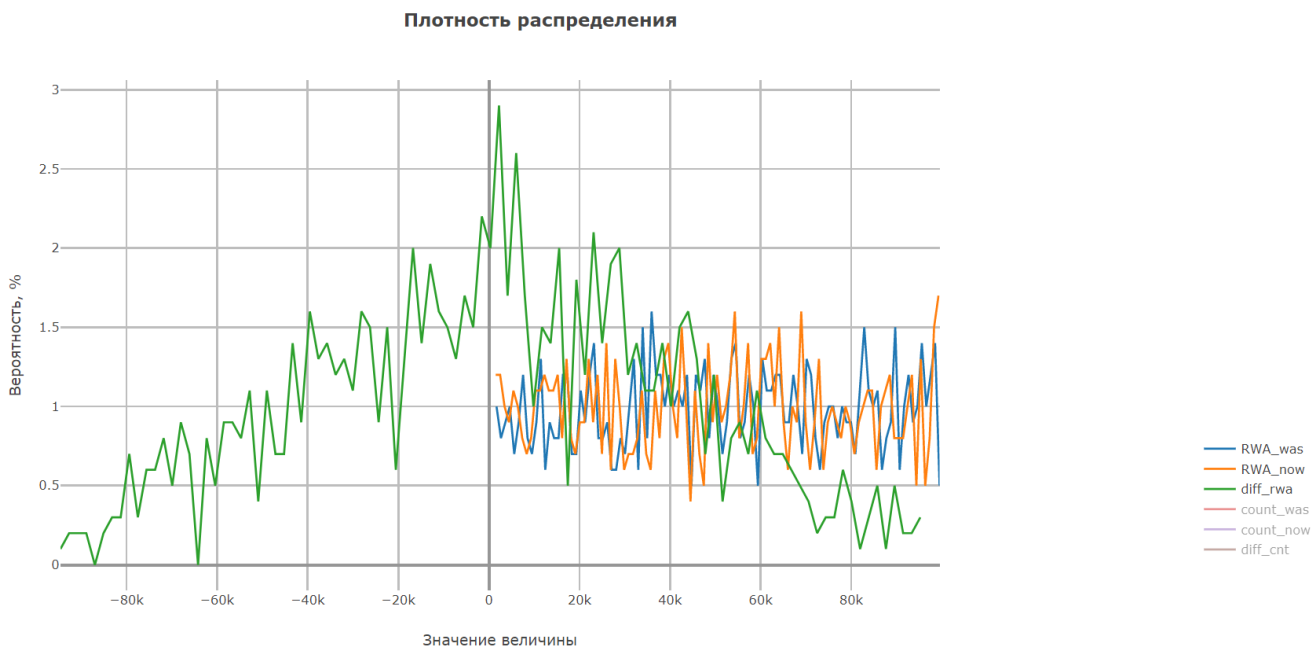
Output (long):

Массив графиков

Output (short):

Отсутствует

Output example (picture):



cd_2_5_Pivot_table_filtered

- **Техническое название**: cd_2_5_Pivot_table_filtered
- **Описание**: Filtered Pivot Table. Сводная таблица, значения фильтруются кнопками
- **Теги**: core, data
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Сводная таблица.
Кнопки фильтруют входной набор данных для расчета по значениям filter_column.
Для каждого столбца из **value_columns** применяются все функции из **agg_funcs**.
ВНИМАНИЕ: применение математических агрегацией вызовет ошибку, если в **value_columns** есть хоть один столбец с нечисловыми данными

Входные параметры

df: Датасет для исследования (dataframe)
filter_column: Столбец, по значениям которого фильтруются данные (column)
index_column: Столбец, значения которого образуют индекс пивот-таблицы (column)
value_columns: Список столбцов для вычисления показателей (sum) по группам (multi-column)
agg_funcs: Список агрегирующих функций, используемых над **value_columns**. Default=['sum'].

Результаты

Движок отрисовки графика: plotly.js

Output (long):

Сводная таблица

Output (short):

Отсутствует

Output example (picture):

Pivot table for PD_Grade
filter: Продукт

All Продукты
Продукт 1
Продукт 2
Продукт 3
Продукт 4
Продукт 5

PD_Grade	RWA_now_sum	RWA_was_sum	count_now_sum	count_was_sum	diff_cnt_sum	diff_rwa_sum
1	2180151.39	2592087.28	2605	2518	87	411935.89
2	3235930.01	2480343.72	2669	2784	-115	-755586.29
3	2981128.51	2665383.7	2798	2852	-54	-315744.81
4	2881943.78	2702409.21	3152	2889	263	-179534.57
5	2496862.66	2147473.49	2128	2090	38	-349389.17
6	2887227.46	2644392.14	2859	2707	152	-242835.32
7	2716831.52	2392976.34	2614	2317	297	-323855.18
8	2311735.35	2627458.48	2713	2693	20	315723.13
9	2589240.97	3274695.48	3412	3105	307	685454.51
10	2237800.09	2989002.89	1989	2771	-782	751202.8
11	2403094.17	2643407.29	2317	2535	-218	240313.12
12	2267538.44	2422741.11	2204	1711	493	155202.67
13	2640276.62	2248313.63	2384	2340	44	-391962.99
14	2980953.26	3200616	2901	2873	28	219662.74
15	2507818.24	2368328.29	2414	2029	385	-139489.95
16	1937310.49	2111594.75	1979	2084	-105	174284.26
17	2181581.73	2509093.33	2551	2577	-26	327511.6
18	2390222.68	1832896.59	2265	2385	-120	-557326.09
19	2479008.11	2894381.34	2599	3096	-497	415373.23
20	2145054.68	2751847.37	2174	2096	78	606792.69

cd_2_6_Barchart_filtered

- **Техническое название**: cd_2_6_Barchart_filtered
- **Описание**: Filtered Barchart. Столбчатая диаграмма, значения фильтруются кнопками
- **Теги**: core, data
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Кнопки фильтруют входной набор данных для расчета по значениям filter_column.
Для столбца **value_column** применяется функция **agg_funcs**.
ВНИМАНИЕ: применение математических агрегацией вызовет ошибку, если **value_column** является столбцом с нечисловыми данными

Входные параметры

df: Датасет для исследования (dataframe)
filter_column: Столбец, по значениям которого фильтруются данные (column)
index_column: Столбец для группировки (column)

value_column: Столбец для вычисления показателей (sum) по группам (column)
agg_func: Список агрегирующих функций, используемых над **value_columns**. Default=['sum'].

Результаты

Движок отрисовки графика: plotly.js

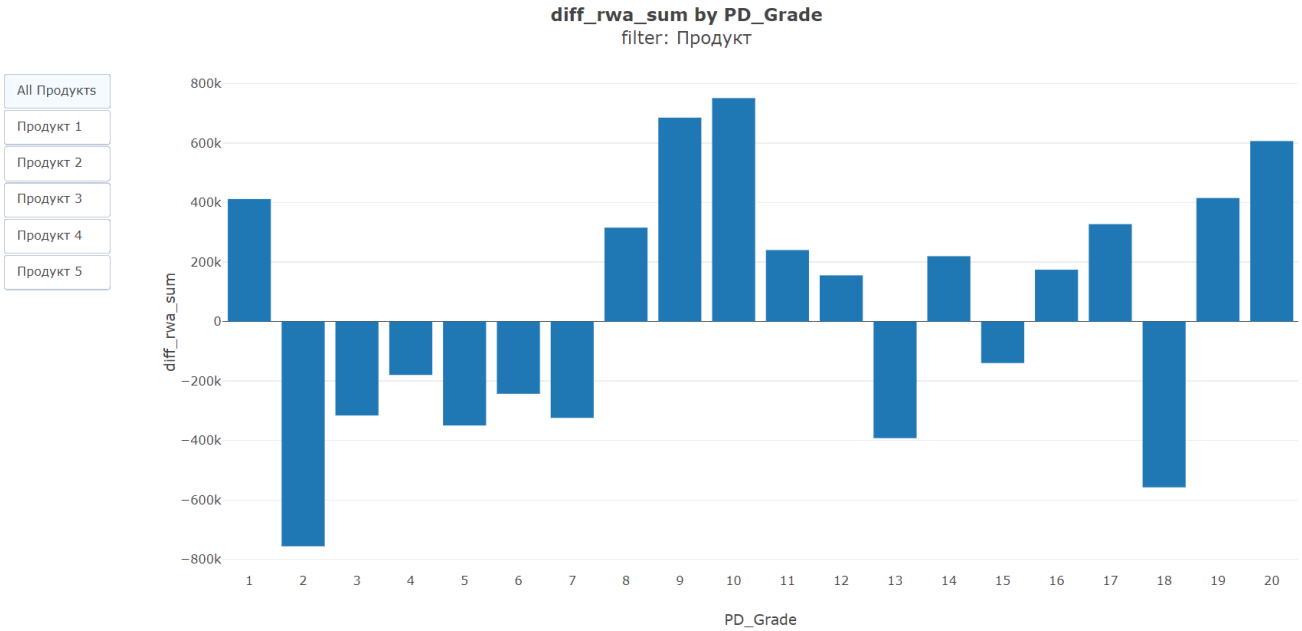
Output (long):

Барчарт
xaxis : значения index_column
yaxis : суммы значений value_column по группам

Output (short):

nan

Output example (picture):



cd_2_7_Fill_Pct

- **Техническое название**: cd_2_7_Fill_Pct
- **Описание**: Fill Percent for Features. Barchart с процентом не-null значений в выбранных столбцах
- **Теги**: core, data
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Для каждого из выбранных столбцов датафрейма вычисляется отношение числа заполненных (не Null) полей к общему числу полей

Входные параметры

df : Объект данных для расчета (dataframe)
field_columns: Фичи для расчета (multi-column)

Результаты

Движок отрисовки графика: plotly.js

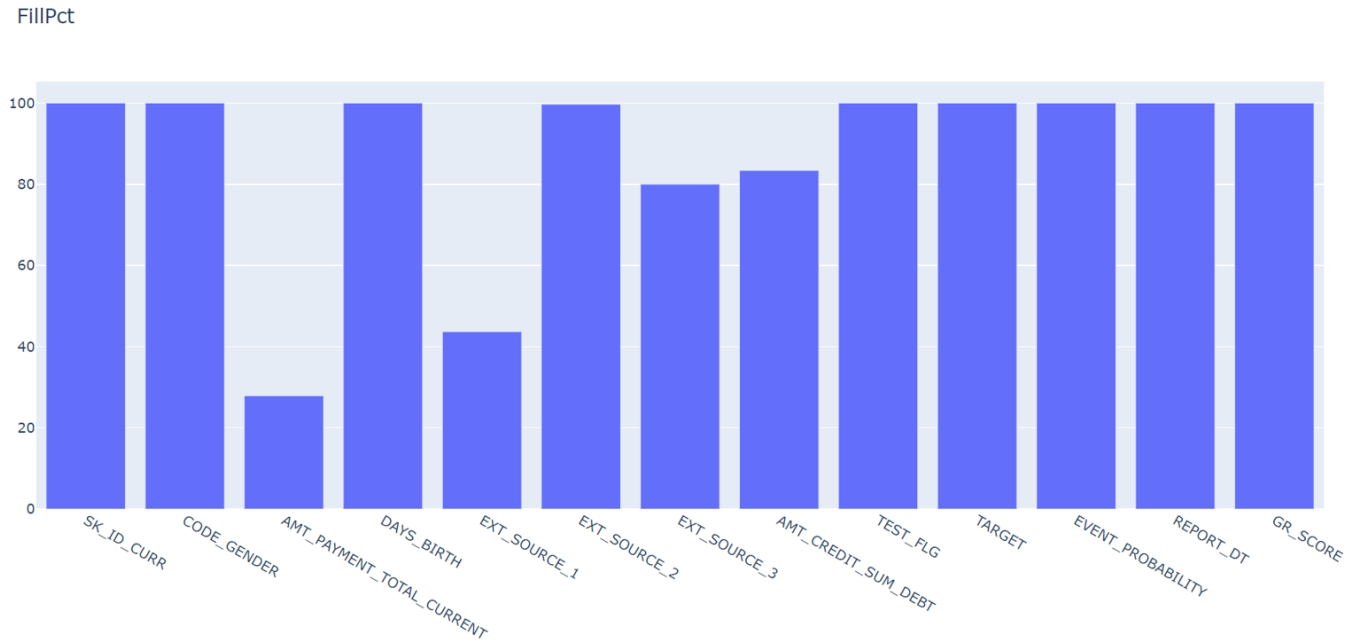
Output (long):

Барчарт
xaxis : названия столбцов, для которых производился расчет
yaxis : процент заполненных значений

Output (short):

Отсутствует

Output example (picture):



АНАЛИЗ РАСПРЕДЕЛЕНИЙ

cd_3_1_Histogram

- **Техническое название**: cd_3_1_Histogram
- **Описание**: Histogram. Гистограмма для выбранного столбца
- **Теги**: core, data
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Дает картину распределения значений в выбранном столбце

Входные параметры

df: Объект данных для расчета (dataframe)
field_column: Столбец для расчета (column)
nbins: Максимальное число столбцов в гистограмме (int value; по умолчанию 30)

Результаты

Движок отрисовки графика: plotly.js

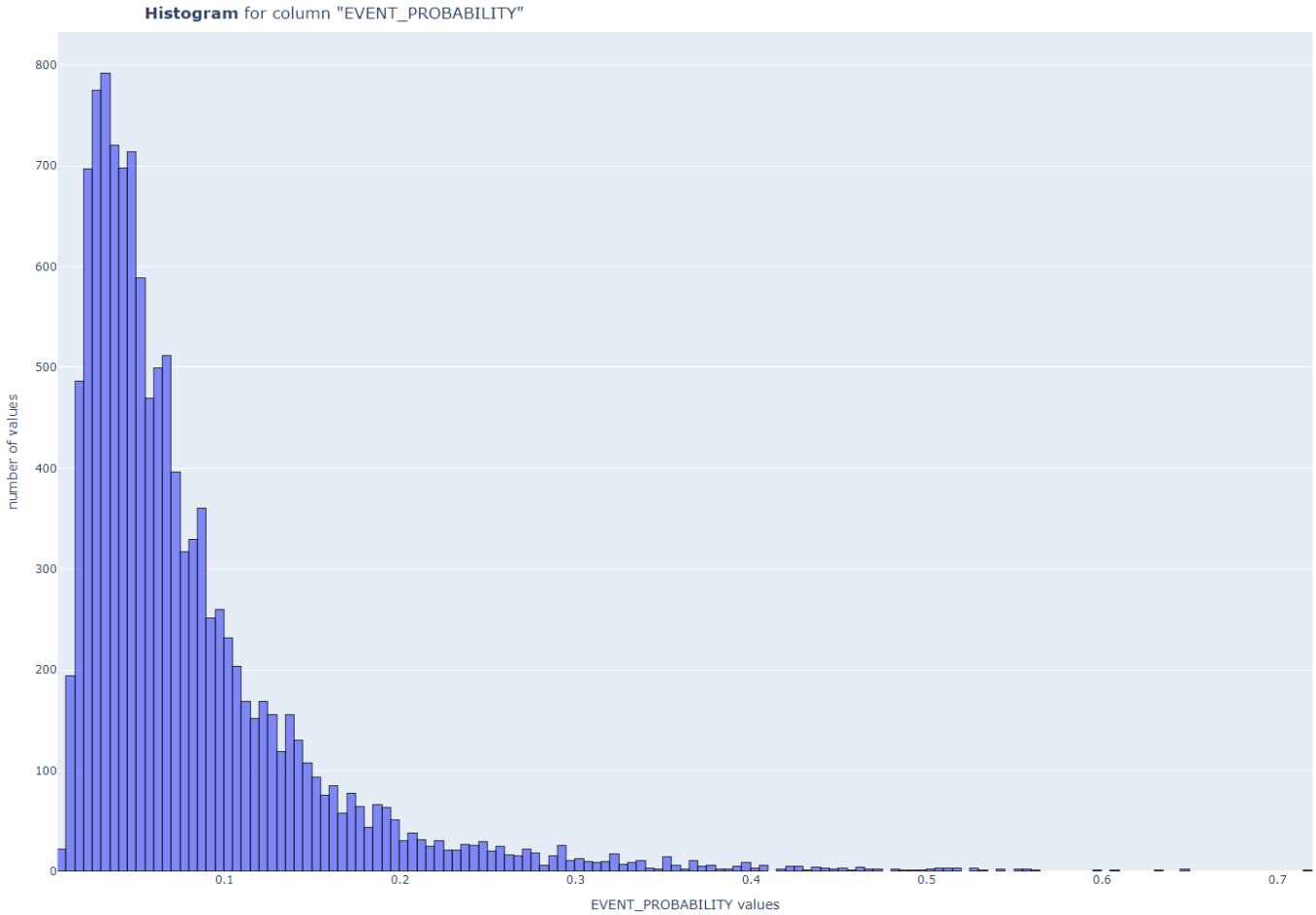
Output (long):

Гистограмма:
xaxis : значения из рассматриваемого столбца
yaxis : частоты встречаемости значений

Output (short):

Отсутствует

Output example (picture):



cd_3_2_Target_Variables_Rates

- **Техническое название**: cd_3_2_Target_Variables_Rates
- **Описание**: Target Variables Rates. Доля целевой переменной по сегментам
- **Теги**: core, data
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Данные разбиваются на группы по значениям cat_field_column. (Число групп = числу уникальных значений cat_field_column)
Для каждой группы определяется:
- доля наблюдений с target_column==1 среди всех наблюдений данной группы
- средний predict_column по данной группе

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: Целевая переменная модели (column)

predict_column:
Скор модели (column)
cat_field_column: Столбец с категориальной переменной, по значениям которой проводим группировку (column)

Результаты

Движок отрисовки графика: plotly.js

Output (long):

Барчарт
xaxis : группы (значения cat-field)
yaxis: средние значения target и score по группам

Output (short):

Отсутствует

Output example (picture):

cd_3_3_Shapiro_Wilk_Test

- **Техническое название**: cd_3_3_Shapiro_Wilk_Test
- **Описание**: Shapiro-Wilk Test. Тест Шапиро-Уилка (нормальность распределения данных)
- **Теги**: core, data, scalar
- **requirements**: `typing`, `pandas`, `scipy.stats`
- **Примечания**: nan

Логика исполнения

Тест на нормальность распределения данных.
H_0: Данная выборка - из нормального распределения.
Вычисляем p-value с помощью scipy.stats.shapiro, по нему настраиваем светофор
Если в столбце есть пропущенные значения, они исключаются из рассмотрения.
Пример выставления signal bounds:
p-value ≤ 1(%) - красный,
1(%) < p-value ≤ 5(%)- желтый,
p-value > 5(%) - зеленый

Входные параметры

df: Объект данных для расчета (dataframe)
field_column: Столбец для расчета (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Число:
значение p-value

Output example (picture): не применимо.

cd_3_4_Percentiles

- **Техническое название**: cd_3_4_Percentiles
- **Описание**: Percentiles. Линейный график в осях номер-значение перцентиля для выбранного столбца
- **Теги**: core, data
- **requirements**: `typing`, `pandas`, `numpy`, `plotly.graph_objects`
- **Примечания**: -

Логика исполнения

Если в столбце есть пропуски, они исключаются из рассмотрения.

Входные параметры

df: датасет с данными для исследования (dataframe)
field_column: название столбца для расчета (column)

Результаты

Движок отрисовки графика: plotly.js

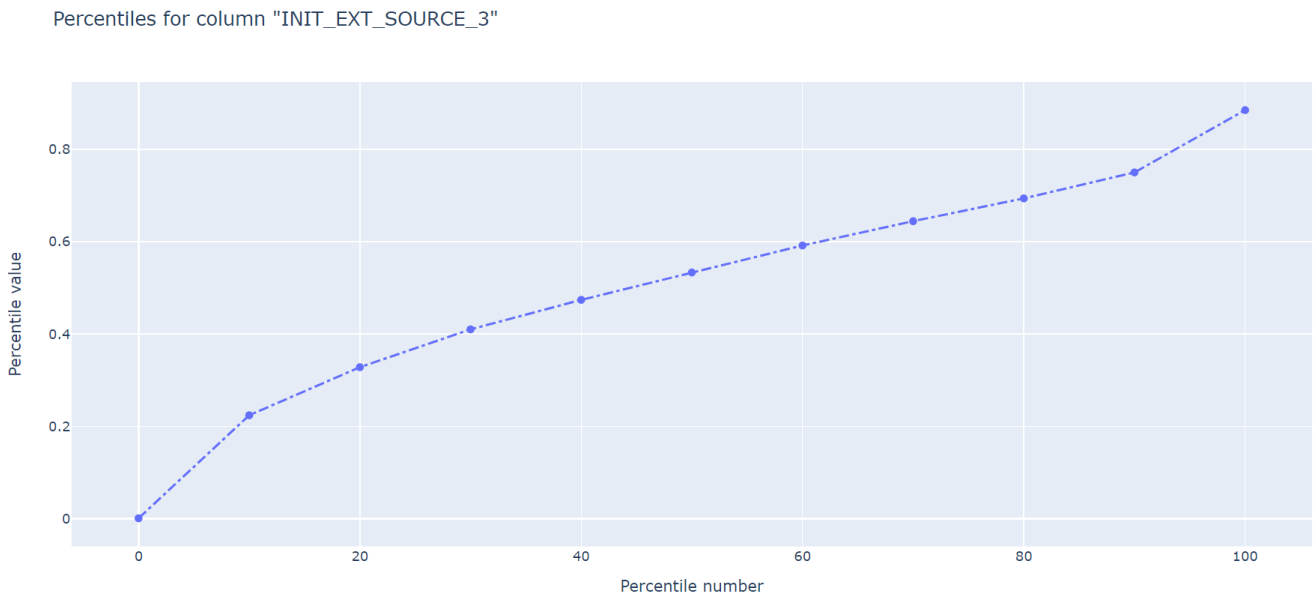
Output (long):

Линейный график
xaxis : номер перцентиля (10, 20, ... 90)
yaxis : значение перцентиля

Output (short):

Отсутствует

Output example (picture):



ПОИСК ЗАВИСИМОСТЕЙ В ДАННЫХ, ОТБОР ПРИЗНАКОВ

cd_4_1_Pearson_Correlations

- **Техническое название**: cd_4_1_Pearson_Correlations
- **Описание**: Pearson Correlations. Корреляции Пирсона (в %)
- **Теги**: core, data, scalar
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

Только для линейных моделей.
Коэффициент корреляции Пирсона характеризует существование линейной связи между двумя признаками (столбцами df).
Пропущенные поля исключаем из рассмотрения.
Пример выставления signal bounds:
max(Corr) > 75- красный,
60 < max(Corr) ≤ 75 - желтый,
max(Corr) ≤ 60 - зеленый

Входные параметры

df: Объект данных для расчета (dataframe)
field_columns:
Фичи для расчета (multi-column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

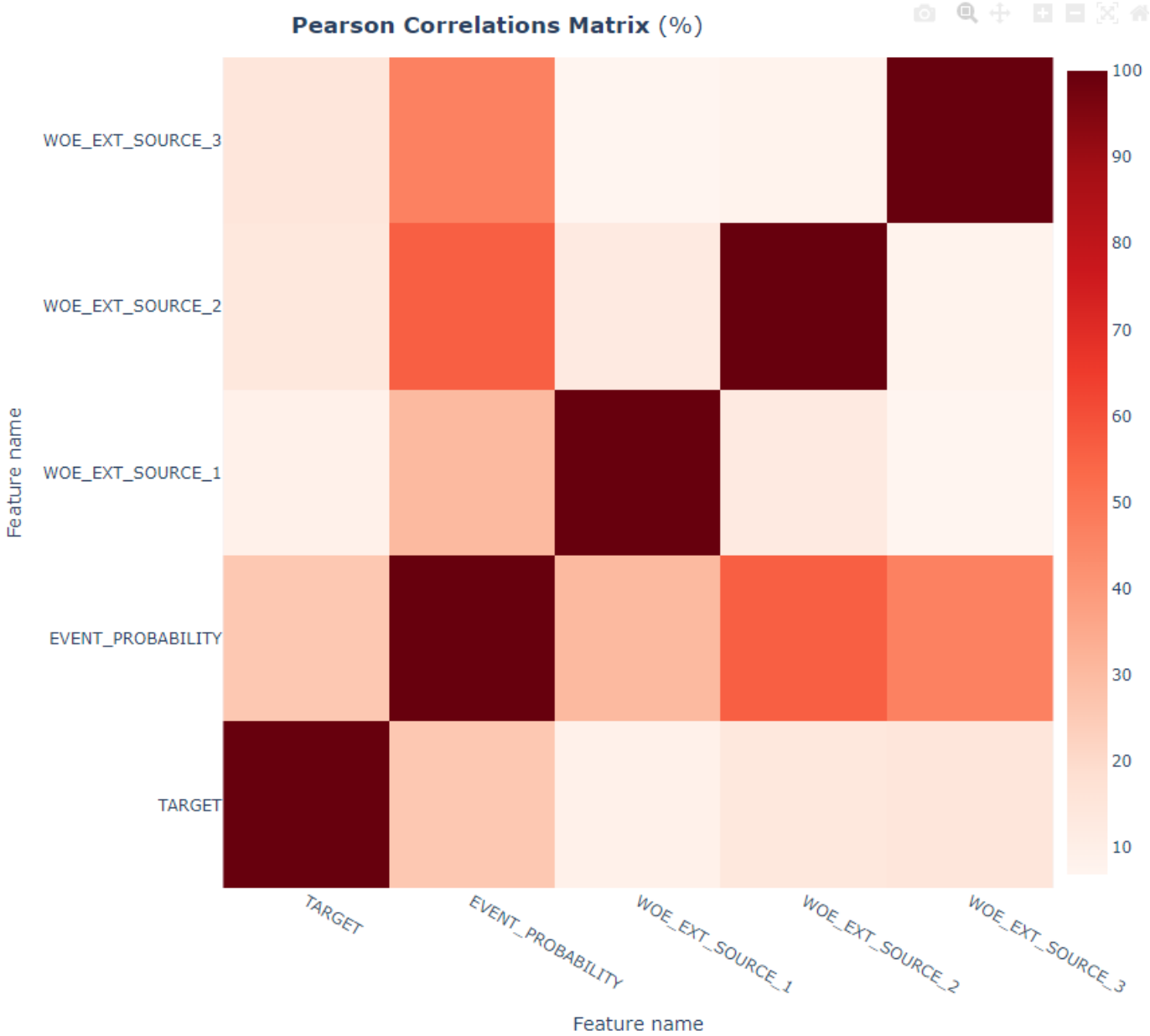
Output (long):

Heatmap
Матрица попарных корреляций выбранных столбцов

Output (short):

Число - максимальное из значений парной корреляции для выбранных столбцов (в %)
Светофор

Output example (picture):



cd_4_2_Gini_features

- **Техническое название**: cd_4_2_Gini_features
- **Описание**: Gini Index (%) for Features. Индекс Джини (%) в разрезе отдельных факторов
- **Теги**: core, data, risk
- **requirements**: `typing`, `pandas`, `sklearn.metrics`
- **Примечания**: -

Логика исполнения

В цикле для каждого field из field_columns:
1. Расчет ROC AUC по target_column, field
2. [Gini index] = 2 * [ROC AUC] - 1
Если в признаке (field) или target_column присутствует пропущенное значение, такая строка исключается из рассмотрения. Для разных признаков

исключаются из рассмотрения разные строки.
Индекс Джини для фактора служит оценкой способности фактора разделять наблюдения разных классов.
Чем больше Джини, тем сильнее по заданному фактору разделяются классы target_column.
Тест используется при отборе факторов для модели (целесообразно выбрать факторы с наибольшими Джини).
При валидации: проверка того, что факторы, используемые в модели, действительно информативные.
Пример выставления signal bounds: Gini ≤ 5(%) - красный,
5(%) < Gini ≤ 15(%) - желтый,
Gini > 15(%) - зеленый

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
field_columns: массив названий столбцов, по которым считаем тест (multi-column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

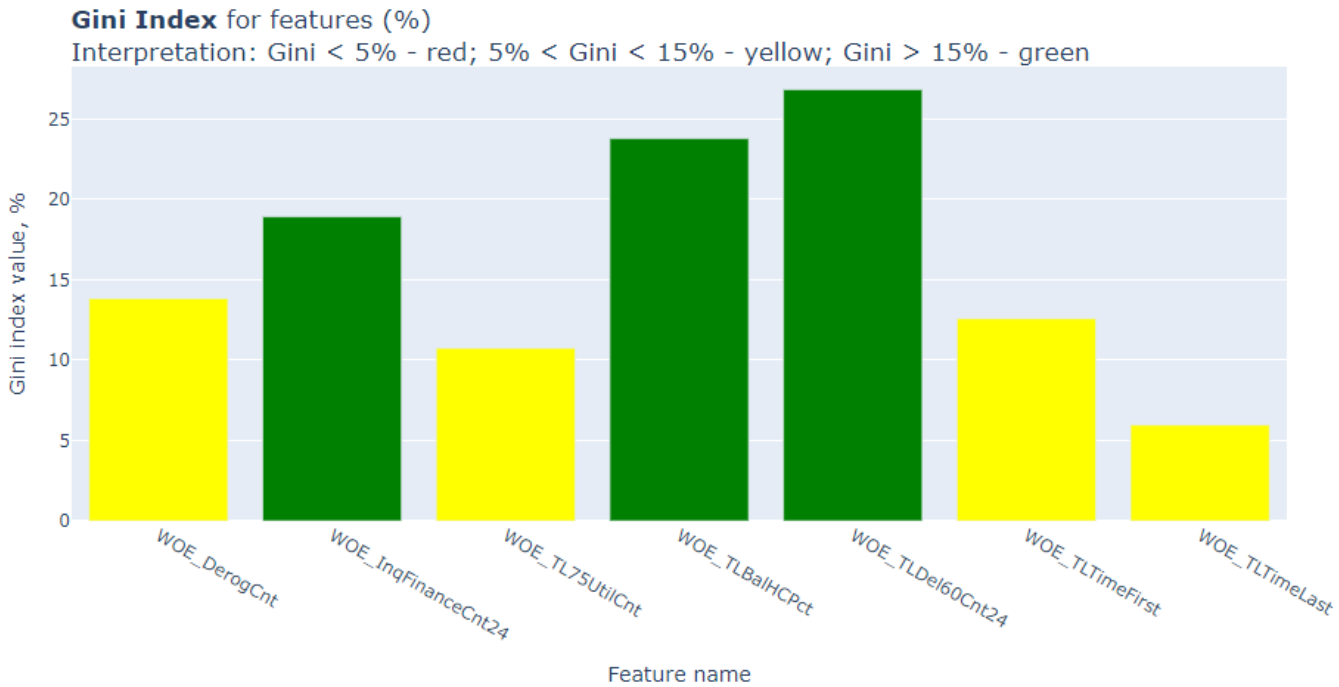
Output (long):

Барчарт:
xaxis : названия столбцов, для которых производился расчет
yaxis : значения коэффициента Джини (в %)
Светофор: столбцы окрашиваются в цвет светофора для соответствующего признака

Output (short):

Отсутствует

Output example (picture):



cd_4_3_Chi_Square_features

- **Техническое название**: cd_4_3_Chi_Square_features
- **Описание**: Chi Square Test for features Pct. Хи квадрат (проверка схожести 2-х категориальных распределений) в процентах

- **Теги**: core, data, scalar
- **requirements**: `typing`, `pandas`, `scipy.stats`
- **Примечания**:
  Применение данной реализации критерия согласия хи-квадрат:
  1) сравнение 2-х категориальных признаков
  2) сравнение распределений target и категориального predict по классам (например, при калибровке модели)

Логика исполнения

В общем случае: тест хи-квадрат проверяет нулевую гипотезу о том, что категориальные данные имеют заданное распределение по группам.
В реализации:
H_0: Распределения первой и второй категориальных переменных одинаковы
Выполнение H_0 - желаемый исход => чем больше p-value, тем лучше
Пример выставления светофора:
p-value ≤ 1(%) - красный,
1(%) < p-value ≤ 5(%)- желтый,
p-value > 5(%) - зеленый

Входные параметры

df: Объект данных для расчета (dataframe)
cat_feature_1_column: Первая категориальная переменная (column)
cat_feature_2_column: Вторая категориальная переменная (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

p-value критерия хи-квадрат,
светофор

Output example (picture): не применимо.

cd_4_4_VIF (r_3_2_VIF)

- **Техническое название**: cd_4_4_VIF (r_3_2_VIF)
- **Описание**: Variance Inflation Factor. Коэффициент инфляции дисперсии.
  Используется для обнаружения мультиколлинеарности.
- **Теги**: core, data, risk
- **requirements**: `typing`, `pandas`, `statsmodels.stats.outliers_influence`
- **Примечания**: -

Логика исполнения

В цикле для каждого категориального field из fields_to_test считаем statsmodels.stats.outliers_influence.variance_inflation_factor
VIF позволяет оценить степень мультиколлинеарности в рамках анализа взаимозависимости факторов модели методом наименьших квадратов.
Индекс показывает насколько велика зависимость фактора от других факторов модели.
(В реализации - зависимость выбранного фактора от других факторов из fields_to_test).
VIF принимает значения от 1 до +∞. Чем больше значение, тем сильнее свидетельство присутствия мультиколлинеарности.
Значение, равное 1, свидетельствует об ортогональности (независимости) независимой переменной остальным переменным модели.
Используется для линейных моделей
Пример выставления signal bounds:

VIF ≥ 5- красный,
3 ≤ VIF < 5- желтый,
VIF < 3 - зеленый

Входные параметры

df: датасет с данными для исследования (dataframe)
field_columns: массив названий столбцов, по которым считаем тест (multi-column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

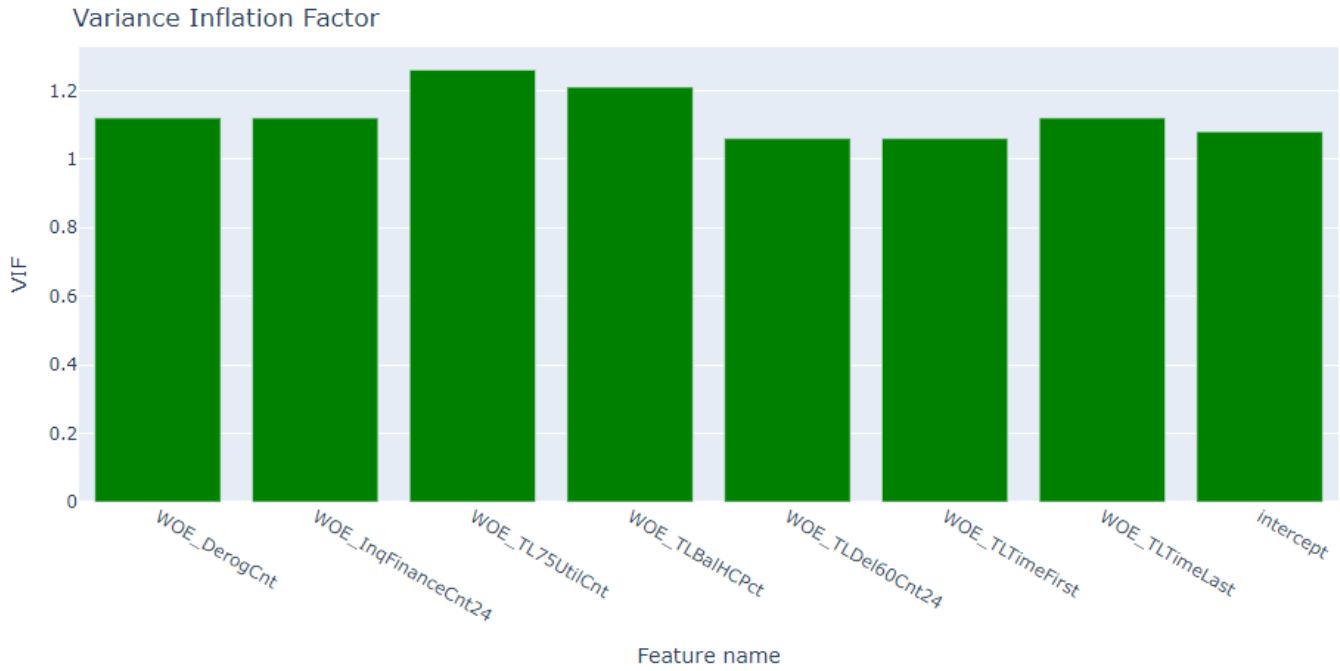
Output (long):

Барчарт:
xaxis : названия столбцов, для которых производился расчет
yaxis : значения VIF
Светофор: столбцы окрашиваются в цвет светофора для соответствующего признака

Output (short):

Отсутствует

Output example (picture):



Core package (базовый функционал). Метрики качества регрессии

СПИСОК МЕТРИК.

rvc_1_MAE

- **Техническое название**: rvc_1_MAE
- **Описание**: Mean Absolute Error (MAE). Средняя абсолютная ошибка
- **Теги**: core, regression, scalar

- **requirements**: `typing`, `pandas`, `sklearn.metrics`
- **Примечания**: Применять только на датасетах с небинарным таргетом

Логика исполнения

Средняя абсолютная ошибка
(Средняя абсолютная разность между предсказаниями модели и целевыми значениями)
Строки с пропусками в target_column или predict_column исключаются из рассмотрения.

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: НЕБИНАРНАЯ
целевая переменная модели (column)
predict_column: Предсказание модели (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Числовое значение средней абсолютной ошибки,
светофор

Output example (picture): не применимо.

rvc_2_RMSE

- **Техническое название**: rvc_2_RMSE
- **Описание**: Root Mean Square Error (RMSE). Среднеквадратичная ошибка
- **Теги**: core, regression, scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`
- **Примечания**: -

Логика исполнения

Корень из среднеквадратичной ошибки
(Квадратный корень из отношения суммы квадратов отклонений предсказаний модели от истинных значений к количеству наблюдений)
Строки с пропусками в target или score исключаются из рассмотрения.

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: Целевая переменная модели (column)
predict_column: Предсказание модели (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Числовое значение среднеквадратичной ошибки,
светофор

Output example (picture): не применимо.

rvc_3_MAPE

- **Техническое название**: rvc_3_MAPE
- **Описание**: Mean Absolute Percentage Error (MAPE). Средняя абсолютная ошибка в процентах
- **Теги**: core, regression, scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`
- **Примечания**: Применять только на датасетах с небинарным таргетом

Логика исполнения

Среднее отношение абсолютной ошибки предсказания к величине предсказанного значения
Строки с пропусками в target_column или predict_column исключаются из рассмотрения.
Также исключаются из рассмотрения строки с target==0, поэтому применение метрики корректно только на датасетах для регрессии (с небинарным таргетом)

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: НЕБИНАРНАЯ
целевая переменная модели (column)
predict_column: Предсказание модели (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Числовое значение MAPE
светофор

Output example (picture): не применимо.

rvc_4_MASE

- **Техническое название**: rvc_4_MASE
- **Описание**: Mean absolute scaled error. Средняя абсолютная масштабированная ошибка
- **Теги**: core, regression, scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`, `numpy`
- **Примечания**: -

Логика исполнения

Средняя абсолютная масштабированная ошибка
В прогнозировании временных рядов MASE дает представление об эффективности алгоритма прогнозирования по отношению к наивному прогнозу. Ее значение больше единицы (1) указывает на то, что алгоритм работает плохо по сравнению с наивным прогнозом.

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: Целевая переменная модели (column)
predict_column: Предсказание модели (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Число:
значение средней абсолютной масштабированной ошибки

Output example (picture): не применимо.

rvc_5_WAPE

- **Техническое название**: rvc_5_WAPE
- **Описание**: Weighted Absolute Percentage Error (WAPE). Взвешенная абсолютная ошибка в процентах
- **Теги**: core, regression, scalar
- **requirements**: -
- **Примечания**: Применять только на датасетах с небинарным таргетом

Логика исполнения

(df[target] - df[predict]).abs().sum() / df[target].abs().sum()
Строки с пропусками в target_column или predict_column исключаются из рассмотрения.

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: НЕБИНАРНАЯ
целевая переменная модели (column)
predict_column: Предсказание модели (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

Output (long):

Отсутствует

Output (short):

Числовое значение WAPE
светофор

Output example (picture): не применимо.

rvc_6_Weighted_MAPE

- **Техническое название**: rvc_6_Weighted_MAPE
- **Описание**: Weighted MAPE (WMAPE). Взвешенная ошибка MAPE

- **Теги**: core, regression, scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`
- **Примечания**: Применять только на датасетах с небинарным таргетом

Логика исполнения

Значения MAPE, взвешенные по столбцу weight_column
Строки с пропусками в target_column, predict_column или weight_column исключаются из рассмотрения.

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: НЕБИНАРНАЯ
целевая переменная модели (column)
predict_column: Предсказание модели (column)
weight_column: Столбец используемый в качестве веса (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

Output (long):

Отсутствует

Output (short):

Число:
значение взвешенной ошибки,
светофор

Output example (picture): не применимо.

rvc_7_Average_Bias

- **Техническое название**: rvc_7_Average_Bias
- **Описание**: Average Bias. Среднее смещение для Squared Loss
- **Теги**: core, regression, scalar
- **requirements**: `typing`, `pandas`, `numpy`
- **Примечания**: -

Логика исполнения

Опр: Матожидание разности между истинным ответом и ответом, выданным алгоритмом
В реализации:
среднее смещение рассчитывается по формуле:
mean(abs(target - mean(score)))
Разложение ошибки на смещение (bias) и разброс (variance): bias_variance_decomp

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: Целевая переменная модели (column)
predict_column: Предсказание модели (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Числовое значение среднего смещения,
светофор

Output example (picture): не применимо.

rvc_8_R_Squared_Score

- **Техническое название**: rvc_8_R_Squared_Score
- **Описание**: R2 score. Коэффициент детерминации
- **Теги**: core, regression, scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`
- **Примечания**: Применять только на датасетах с небинарным таргетом

Логика исполнения

R2 используется для оценки производительности модели машинного обучения на основе регрессии. Суть его работы заключается в измерении количества отклонений в прогнозах, объясненных набором данных (разница между выборками в наборе данных и прогнозами, сделанными моделью).

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: НЕБИНАРНАЯ
целевая переменная модели (column)
predict_column: Предсказание модели (column)
weight_column: Столбец используемый в качестве веса (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Числовое значение коэффициента детерминации,
светофор

Output example (picture): не применимо.

rvc_9_Regression_Performance

- **Техническое название**: rvc_9_Regression_Performance
- **Описание**: Regression Performance. Распределение ошибок регрессии и диаграммы рассеяния в осях актуальные-предсказанные значения для обучающих и текущих данных
- **Теги**: core, regression
- **requirements**: `typing`, `pandas`
- **Примечания**: Для больших датасетов получается очень объемный json графика (что сказывается на скорости отображения результата в UI). Подумать как выводить не все точки, при этом верно отображая вид зависимостей.

Логика исполнения

1) Гистограммы распределения ошибки предсказания (для 2-х датафреймов);
2) Scatter plots в координатах Actual value - Predicted value (для 2-х датафреймов).

Входные параметры

df_reference: датасет с данными на базовый период, либо за предыдущий период (dataframe)
df_current: датасет с данными на текущий период (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца с предсказаниями модели (column)
(! названия столбцов с target_column и predict_column должны совпадать в df_reference и df_current)

Результаты

Движок отрисовки графика: plotly.js

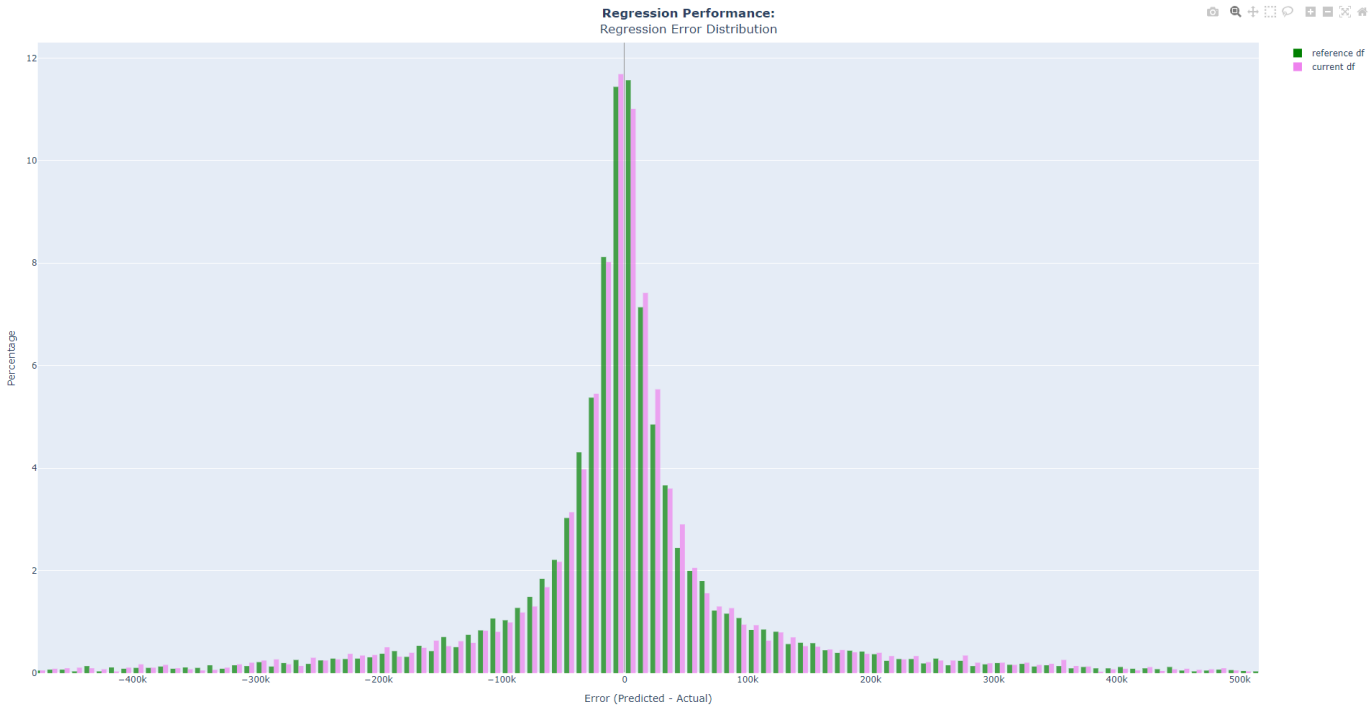
Output (long):

Распределение ошибок регрессии:
1. Гистограмма ошибок на df_reference
2. Гистограмма ошибок на df_current
Диаграммы рассеяния в осях актуальные-предсказанные значения:
1. Диаграмма значений на df_reference
2. Диаграмма значений на df_current

Output (short):

Отсутствует

Output example (picture):



rvc_10_Reg_Error_Analysis

- **Техническое название**: rvc_10_Reg_Error_Analysis
- **Описание**: Regression Error Analysis. Распределение ошибок регрессии и диаграммы рассеяния в осях актуальные-предсказанные значения для одного датасета
- **Теги**: core, regression
- **requirements**: `typing`, `pandas`
- **Примечания**: -

Логика исполнения

1) Гистограмма распределения ошибки предсказания;
2) Scatter plot в координатах Actual value - Predicted value.

Метрика аналогична п.9, но предназначена для одного датасета

Входные параметры

df: датасет с данными (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца с предсказаниями модели (column)

Результаты

Движок отрисовки графика: plotly.js

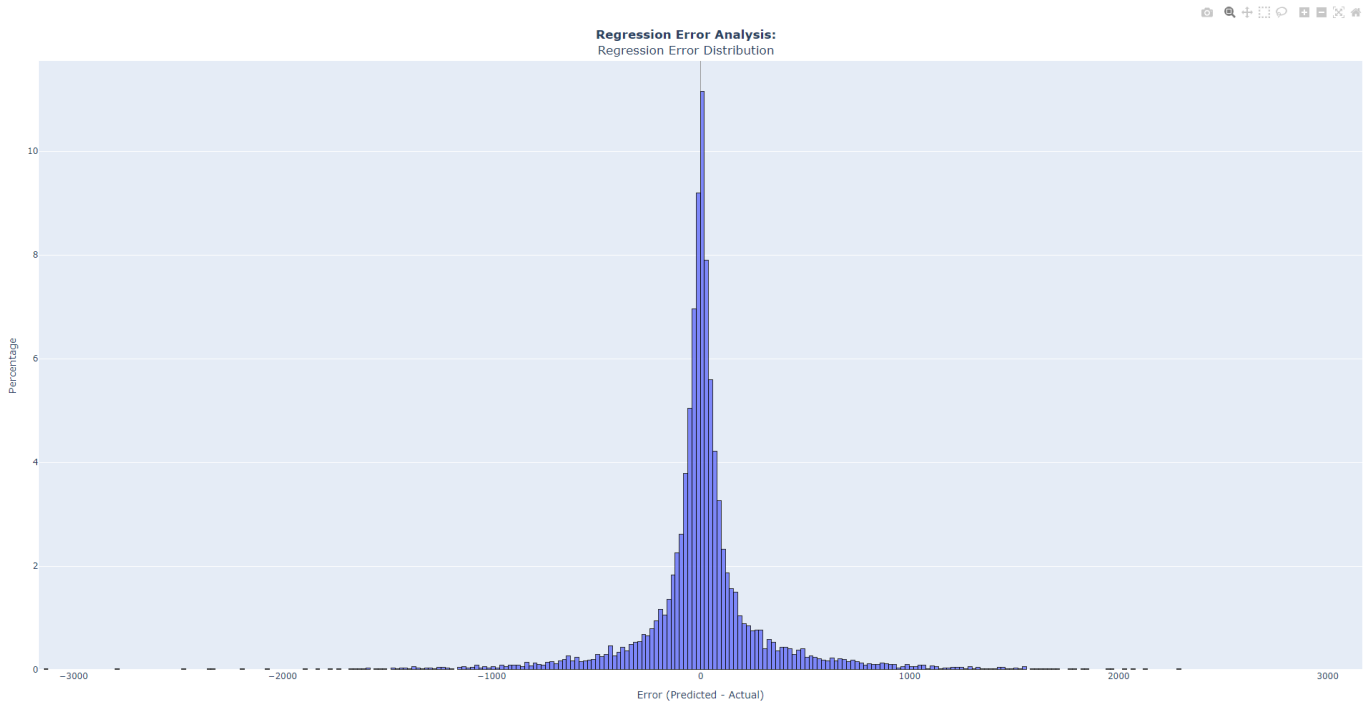
Output (long):

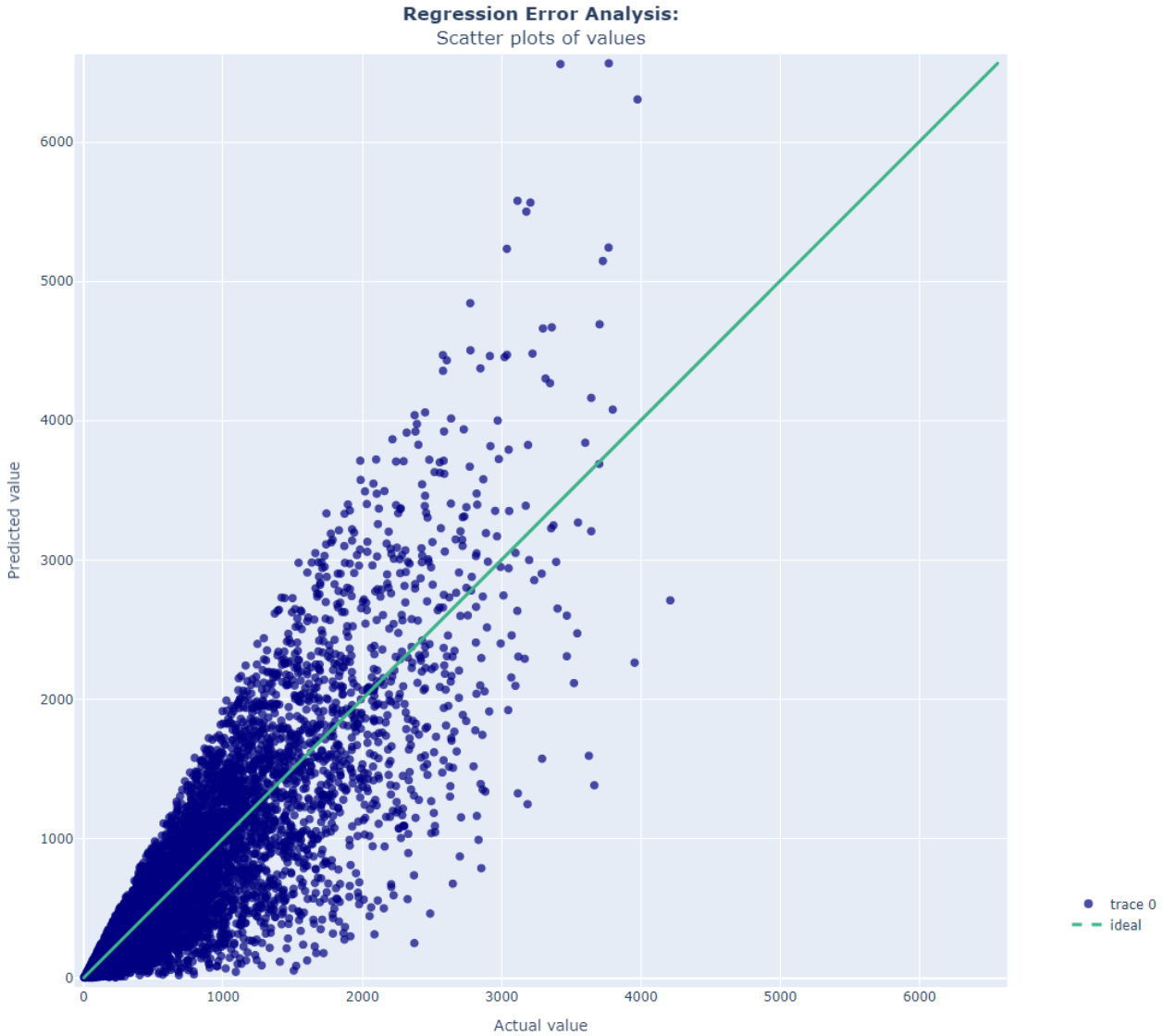
Распределение ошибок регрессии в виде гистограммы
Диаграммы рассеяния в осях актуальные-предсказанные значения

Output (short):

Отсутствует

Output example (picture):





Core package (базовый функционал). Метрики качества классификации

ОЦЕНКА КАЧЕСТВА БИНАРНОЙ КЛАССИФИКАЦИИ

cvc_1_1_F1_score

- **Техническое название**: cvc_1_1_F1_score
- **Описание**: F1 Score. Оценка F1
- **Теги**: core, classification, scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`
- **Примечания**: Реализация может быть доработана для мультиклассовой классификации

Логика исполнения

Показатель, используемый для измерения точности модели бинарной классификации.
Вычисляется по формуле:
F1 = 2 * (precision * recall) / (precision + recall)

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: БИНАРНАЯ
целевая переменная модели (column)
predict_column: БИНАРНОЕ предсказание модели (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Числовое значение оценки F1,
светофор

Output example (picture): не применимо.

cvc_1_2_Precision

- **Техническое название**: cvc_1_2_Precision
- **Описание**: Precision score. Оценка точности
- **Теги**: core, classification, scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`
- **Примечания**: Реализация может быть доработана для мультиклассовой классификации

Логика исполнения

Показатель, используемый для измерения точности модели бинарной классификации.
Вычисляется как доля истинных положительных предсказаний среди всех предсказанных положительных случаев.

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: БИНАРНАЯ
целевая переменная модели (column)
predict_column: БИНАРНОЕ предсказание модели (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Числовое значение точности,
светофор

Output example (picture): не применимо.

cvc_1_3_Recall

- **Техническое название**: cvc_1_3_Recall
- **Описание**: Recall score. Оценка полноты

- **Теги**: core, classification, scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`
- **Примечания**: Реализация может быть доработана для мультиклассовой классификации

Логика исполнения

Показатель, используемый для измерения полноты модели бинарной классификации.
Вычисляется по формуле:
Вычисляется как доля истинных положительных предсказаний среди всех фактических положительных случаев.

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: БИНАРНАЯ
целевая переменная модели (column)
predict_column: БИНАРНОЕ предсказание модели (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Числовое значение полноты,
светофор

Output example (picture): не применимо.

cvc_1_4_Chi_Square_Binary

- **Техническое название**: cvc_1_4_Chi_Square_Binary
- **Описание**: Chi Square Test for binary model. Хи квадрат для модели бинарной классификации
- **Теги**: core, classification, scalar
- **requirements**: `typing`, `pandas`, `scipy.stats`
- **Примечания**: Реализация может быть доработана для мультиклассовой классификации

Логика исполнения

В общем случае: тест хи-квадрат проверяет нулевую гипотезу о том, что категориальные данные имеют заданное распределение по группам.
В реализации:
H_0: Распределения target и predict по классам (категориям target) одинаковы
Если в переменную predict_column подается скор (а не бинарный предикт), предикт вычисляется путем сравнения скора с заданным пользователем значением class_threshold.
Выполнение H_0 - желаемый исход => чем больше p-value, тем лучше
Пример выставления signal bounds:
p-value ≤ 1(%) - красный,
1(%) < p-value ≤ 5(%)- желтый,
p-value > 5(%) - зеленый
Применение критерия согласия хи-квадрат:
1) сравнение распределения средних значений target и predict по группам (например, разрядам рейтинговой шкалы)
2) сравнение распределений target и predict по классам (например, при калибровке модели)

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: БИНАРНАЯ
целевая переменная модели (column)
predict_column: Предсказания модели - в виде вероятностей или бинарные (column)
class_threshold: Порог отсечения классов (float-value, 0.5 по умолчанию)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

p-value критерия хи-квадрат,
светофор

Output example (picture): не применимо.

cvc_1_5_Confusion_Matrix

- **Техническое название**: cvc_1_5_Confusion_Matrix
- **Описание**: Confusion Matrix. Матрица ошибок
- **Теги**: core, classification
- **requirements**: `typing`, `pandas`
- **Примечания**: Реализация может быть доработана для мультиклассовой классификации

Логика исполнения

Используется для оценки точности модели в задаче классификации.
Цветом отображается количество True Positive (TP), False Positive (FP), False Negative (FN), True Negative (TN) решений алгоритма.
Если в переменную score подается именно скор (а не бинарный предикт), предикт вычисляется путем сравнения скора с заданным пользователем значением class_threshold.
(Пример вычисления оптимального порога отсечения в зависимости от выборки см. напр в коде метрики r_4_6 Lift_dynamic)

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: БИНАРНАЯ
целевая переменная модели (column)
predict_column: Предсказания модели - в виде вероятностей или бинарные (column)
class_threshold: Порог отсечения классов (float-value, 0.5 по умолчанию)

Результаты

Движок отрисовки графика: plotly.js

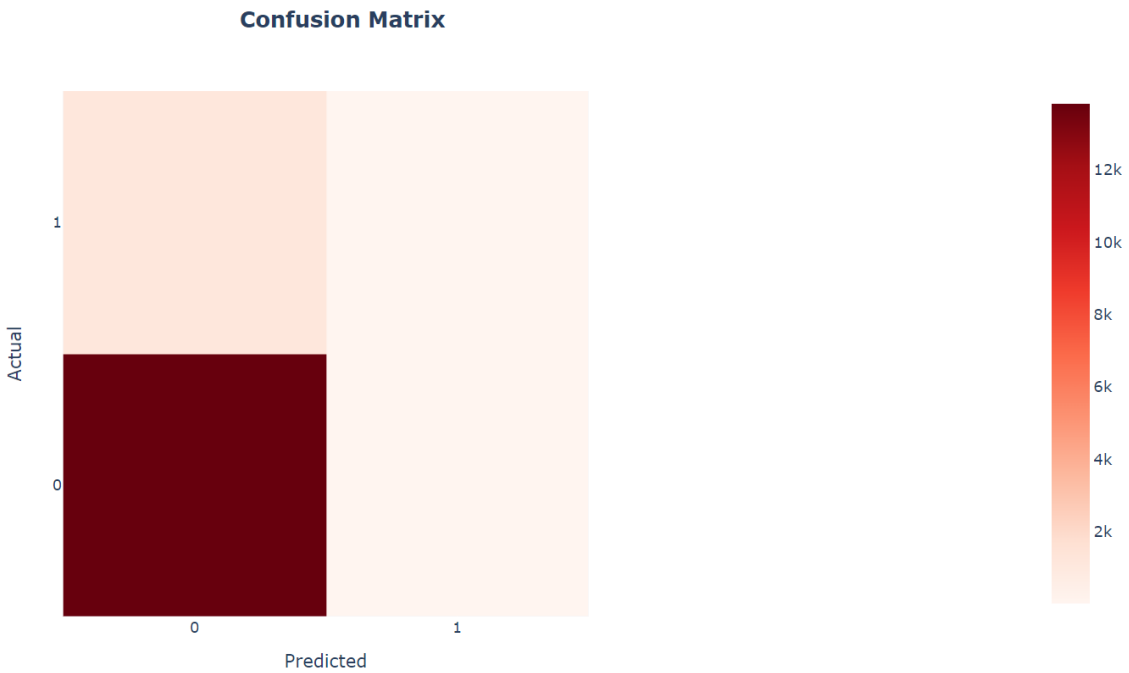
Output (long):

Heatmap
Всего 4 поля:
1 строка - FN, TP
2 строка - TN, FP
При наведении мышки отображается точное количество объектов в каждой категории.

Output (short):

Отсутствует

Output example (picture):



ОЦЕНКА КАЧЕСТВА ВЕРОЯТНОСТНОЙ КЛАССИФИКАЦИИ

cvc_2_1_ROC_AUC

- **Техническое название**: cvc_2_1_ROC_AUC
- **Описание**: График ROC Curve,
  значение ROC-AUC
- **Теги**: core, classification, scalar
- **requirements**: `typing`, `pandas`, `numpy`, `sklearn.metrics`
- **Примечания**: -

Логика исполнения

ROC-кривая используется для оценки качества бинарной классификации, а также вероятностной классификации, для которой производится автоматическое определение порогового значения.
FPR (False Positive Rate) - доля объектов, ошибочно отнесенных алгоритмом к классу 1, среди всех объектов класса 0
TPR (True Positive Rate) - доля верно классифицированных алгоритмом объектов класса 1 среди всех объектов класса 1
AUC = Area Under Curve
Величина ROC AUC (площадь под ROC кривой) характеризует качество классификатора. Чем выше (ближе к 1) показатель AUC, тем качественнее классификатор, Значение 0,5 соответствует случайному классификатору - т.е. модели с ROC AUC < 0.5 бесполезны.

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: Целевая переменная модели (column)
predict_column: Предсказание модели (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

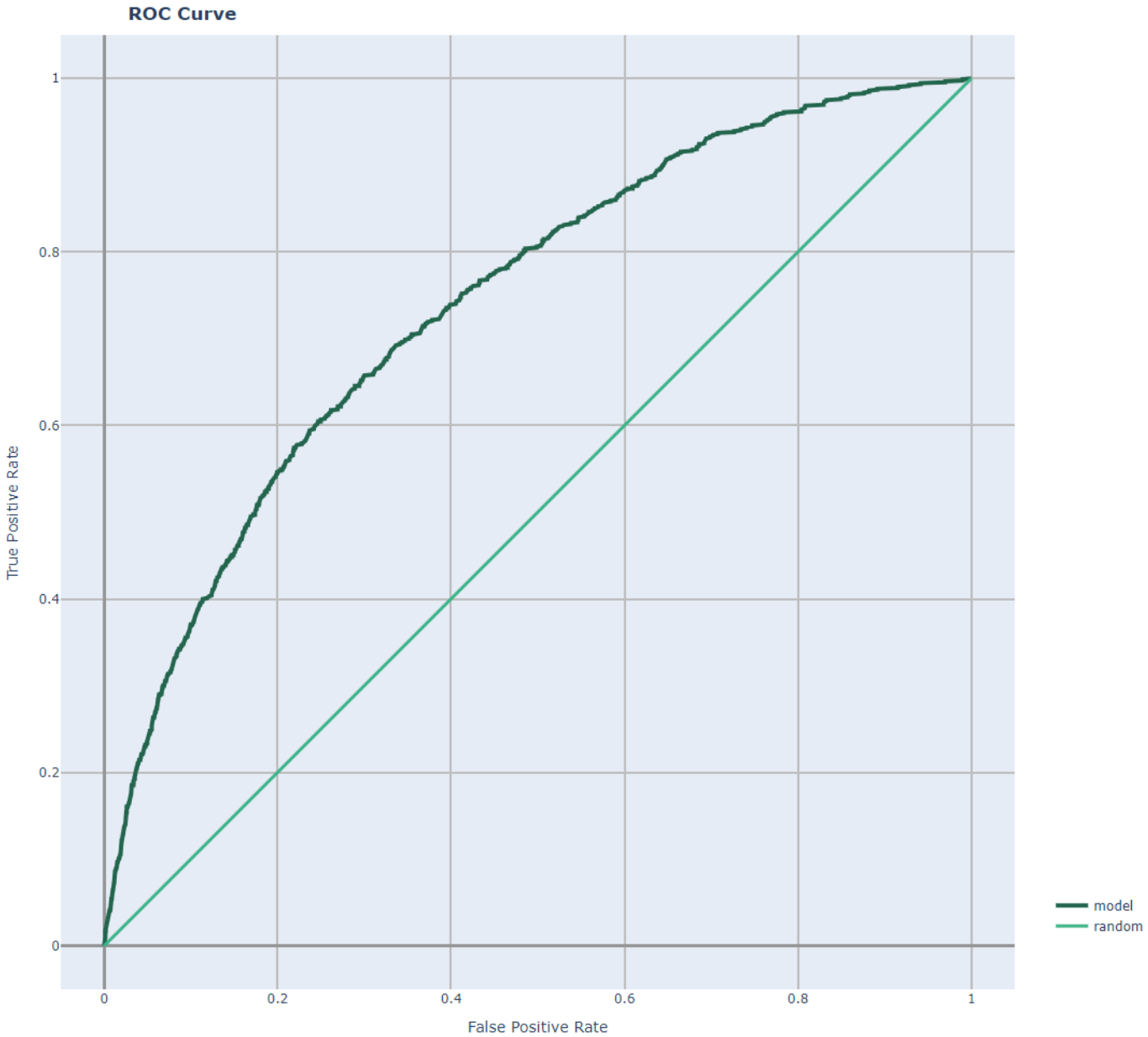
Output (long):

2 линейных графика:
ROC curve для модели,
ROC curve для случайного классификатора
xaxis : FPR
yaxis : TPR

Output (short):

Числовое значение ROC_AUC,
Светофор

Output example (picture):



cvc_2_2_Gain_Curve

- **Техническое название**: cvc_2_2_Gain_Curve
- **Описание**: График Cumulative Gain Curve
- **Теги**: core, classification
- **requirements**: `typing`, `pandas`, `numpy`

- **Примечания**: -

Логика исполнения

Оценка модели классификации, рассчитанная путем соотношения результатов, полученных с моделью и без нее.
Percentage of sample - доля пройденного семпла с предсказаниями, отсортированного от самых высоких предсказаний алгоритма
Percentage of positive target - доля объектов, действительно относящихся к целевому классу, в данной доле семпла

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: Целевая переменная модели (column)
predict_column: Предсказание модели (column)

Результаты

Движок отрисовки графика: plotly.js

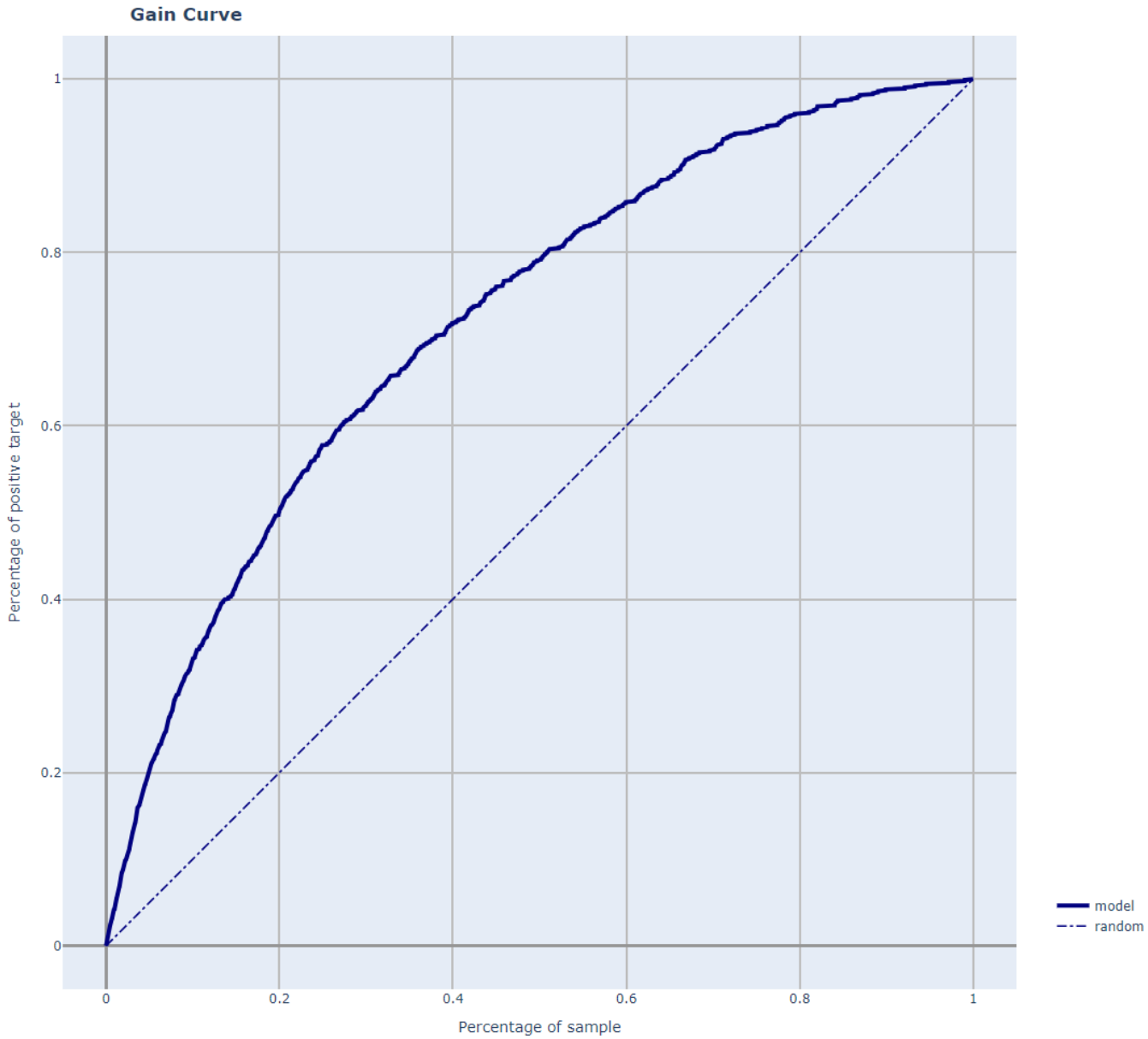
Output (long):

Линейный график
xaxis : Percentage of sample
yaxis : Percentage of positive target

Output (short):

Отсутствует

Output example (picture):



cvc_2_3_Lift_Curve_Cumulative

- **Техническое название**: cvc_2_3_Lift_Curve_Cumulative
- **Описание**: График Lift Curve и Значение Cumulative Lift для топ n% наблюдений
- **Теги**: core, classification, scalar
- **requirements**: `typing`, `pandas`, `numpy`
- **Примечания**: -

Логика исполнения

Отношение высот Gain-кривой и диагонали.
lift = [доля наблюдений с predict_field=1] / [доля наблюдений с target_field=1]
lift = TPR/PR
Интерпретация lift curve:
https://towardsdatascience.com/the-lift-curve-unveiled-998851147871
Обычно Cumulative Lift рассчитывают для топ 10% или топ 30% наблюдений.
Cumulative Lift убывает с ростом n.

Чем больше значение Cumulative Lift, тем лучше модель.
Пример выставления signal bounds для n=10: Cumul Lift ≤ 3 - красный,
3< Cumul Lift ≤ 3.5 - желтый,
Cumul Lift > 3.5 - зеленый

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: Целевая переменная модели (column)
predict_column: Предсказание модели (column)
n_perc: Процент наблюдений для расчета Cumulative Lift (int value; по умолчанию 10)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

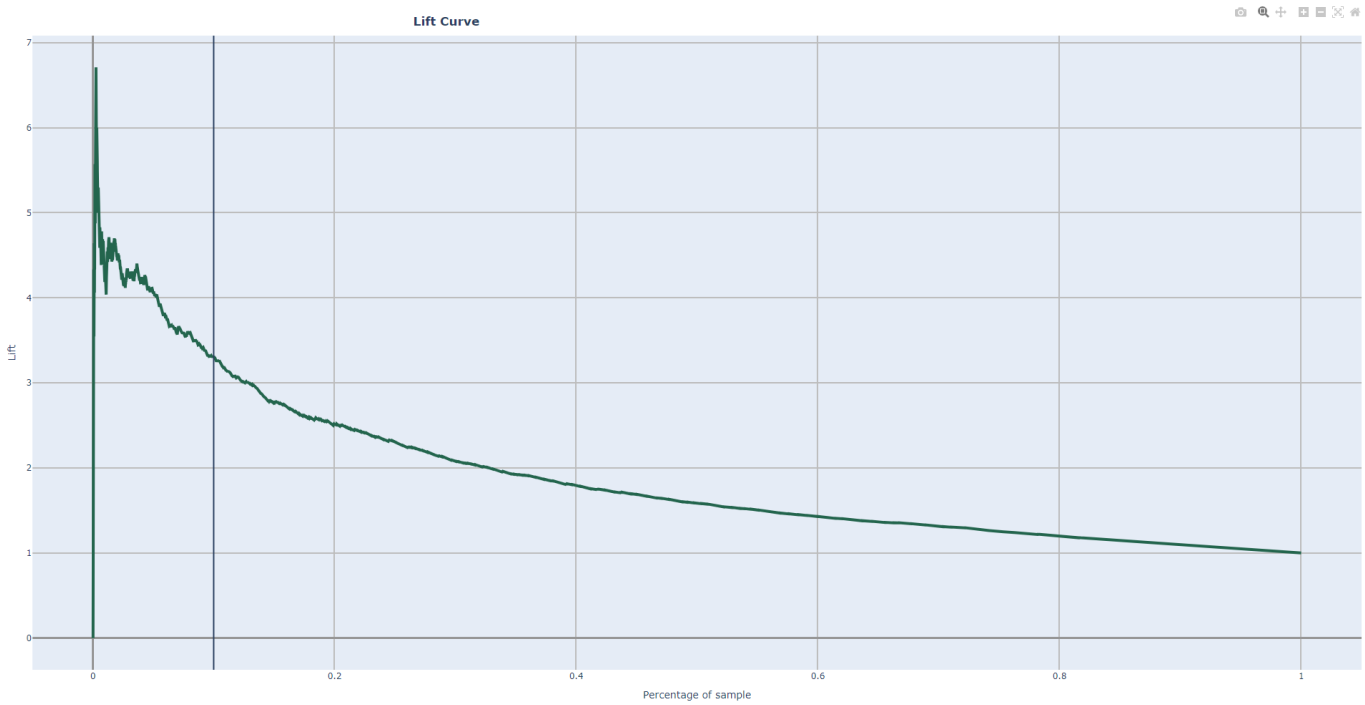
Output (long):

Линейный график
xaxis : PR
yaxis : lift =TPR/PR

Output (short):

Значение Cumulative Lift для топ n% наблюдений,
Светофор

Output example (picture):



cvc_2_4_CAP_Curve_accuracy_rate

- **Техническое название**: cvc_2_4_CAP_Curve_accuracy_rate
- **Описание**: CAP Curve and AR score. График Cumulative Accuracy Profile и значение accuracy rate (%)
- **Теги**: core, classification, scalar
- **requirements**: `typing`, `pandas`, `numpy`, `sklearn.metrics`
- **Примечания**: -

Логика исполнения

CAP curve - это аналог Gain curve (см. 3.4)
Оценка эффективности модели классификации, рассчитанная путем соотношения результатов, полученных с моделью и без нее.
PR(Positive Rate) - доля объектов, отнесенных алгоритмом к классу 1, среди всех объектов
TPR(True Positive Rate) - доля верно классифицированных алгоритмом объектов класса 1 среди всех объектов класса 1

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: Целевая переменная модели (column)
predict_column: Предсказание модели (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

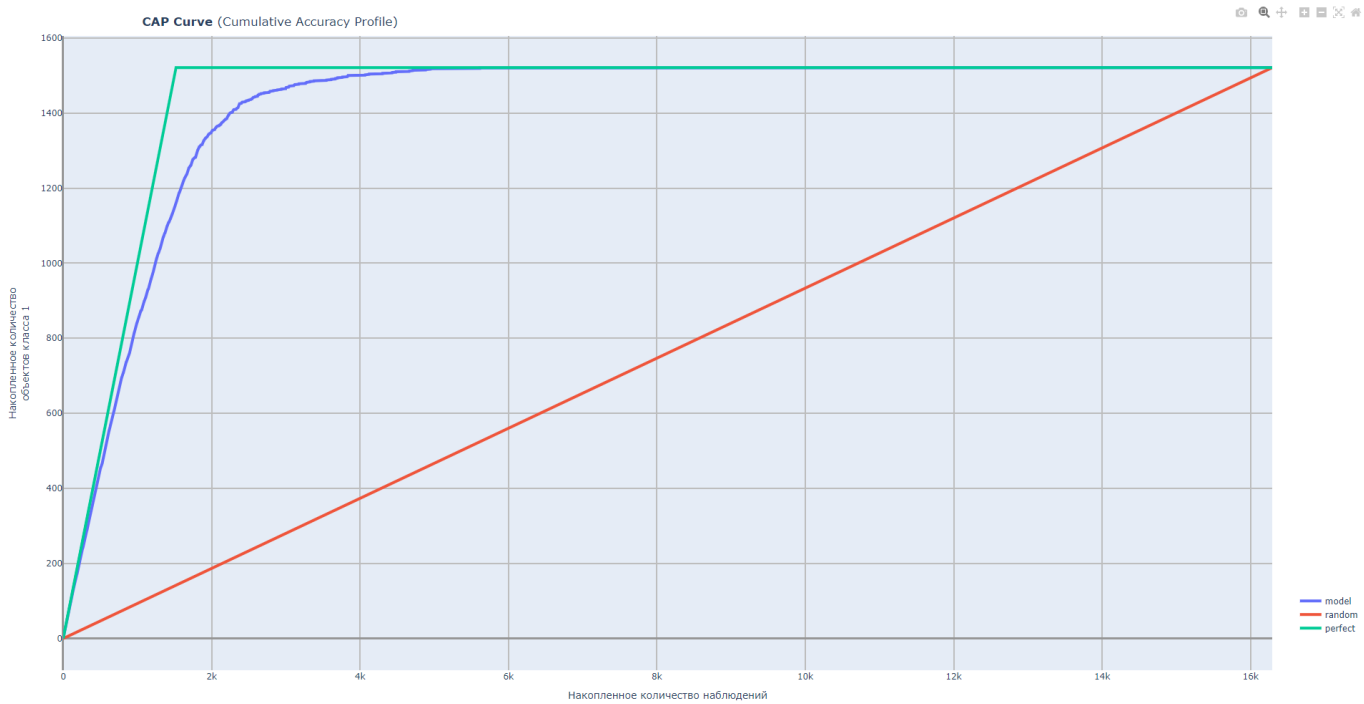
Output (long):

Линейный график
xaxis : PR
yaxis : TPR

Output (short):

Число:
значение accuracy rate на основе CAP по модели (в %)
Светофор

Output example (picture):



cvc_2_5_PR_Curve_PRC_AUC

- **Техническое название**: cvc_2_5_PR_Curve_PRC_AUC
- **Описание**: PR Curve and PRC-AUC. График Precision-Recall Curve и Значение PRC-AUC
- **Теги**: core, classification, scalar

- **requirements**: `typing`, `pandas`, `numpy`, `sklearn.metrics`
- **Примечания**: -

Логика исполнения

График отношения recall и precision с с разными границами.

Входные параметры

df: Объект данных для расчета (dataframe)
target_column: Целевая переменная модели (column)
predict_column: Предсказание модели (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

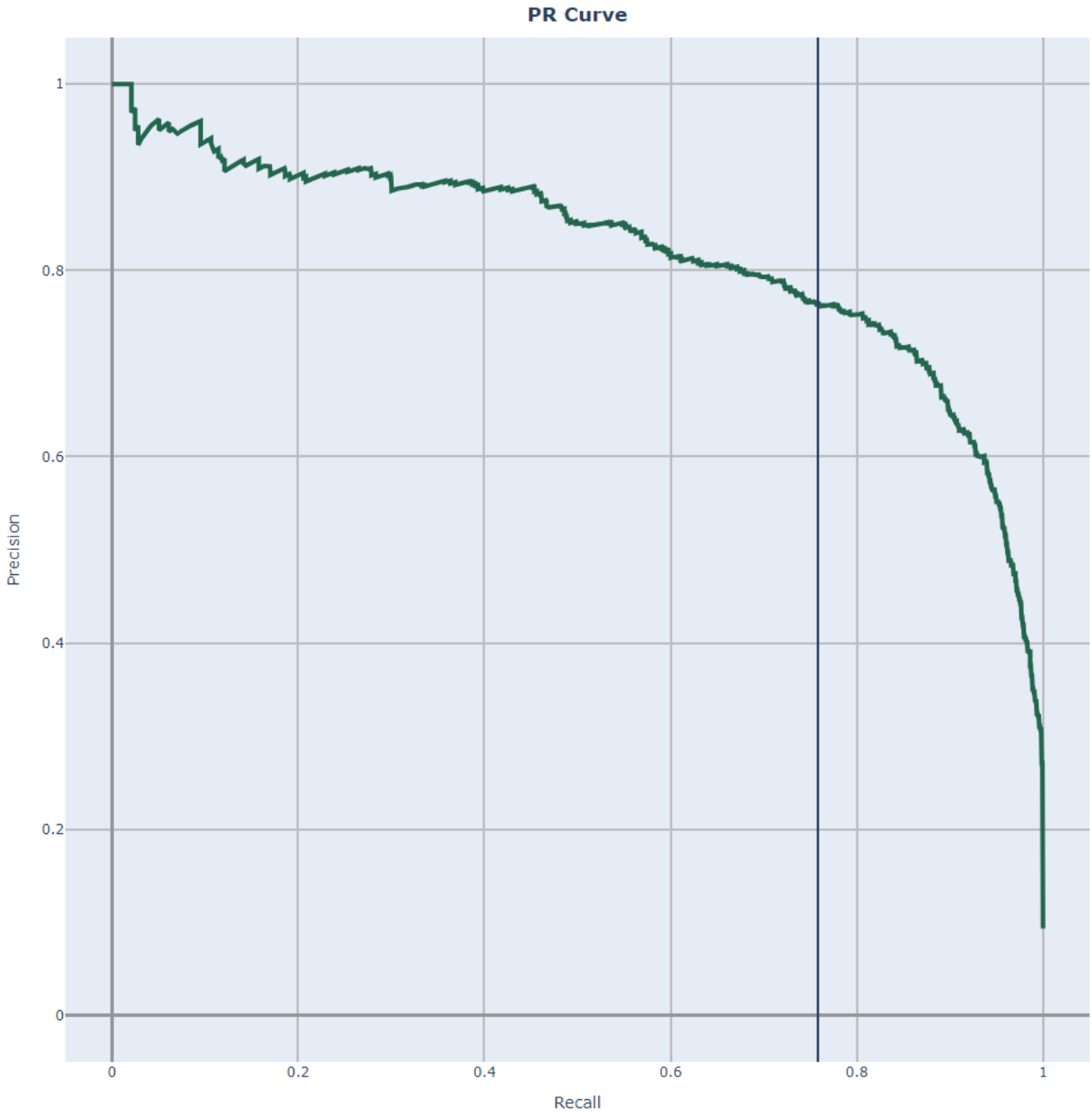
Output (long):

Линейный график:
PR Curve
xaxis : recall (полнота)
yaxis: precision (точность)
Вертикальная линия - оптимальная граница для значений predict_column

Output (short):

Числовое значение PRC-AUC,
Светофор

Output example (picture):



cvc_2_6_AUCs_Dynamic

- **Техническое название**: cvc_2_6_AUCs_Dynamic
- **Описание**: AUCs Dynamic. Динамика ROC AUC и PRC AUC модели на исторических данных
- **Теги**: core, classification
- **requirements**: `typing`, `pandas`, `sklearn.metrics`
- **Примечания**: -

Логика исполнения

Наблюдения из df разбиваются на группы по значениям столбца с датой (report_dt). В одну группу попадают все наблюдения за period (месяцы, квартал или год).
Для наблюдений каждой группы рассчитываются ROC AUC и PRC AUC.

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
report_dt_column: название столбца с датой (column)
period: гранулярность данных
(dropdown - одно из значений типа str: 'month', 'quarter', 'year';
по умолчанию -'quarter')

Результаты

Движок отрисовки графика: plotly.js

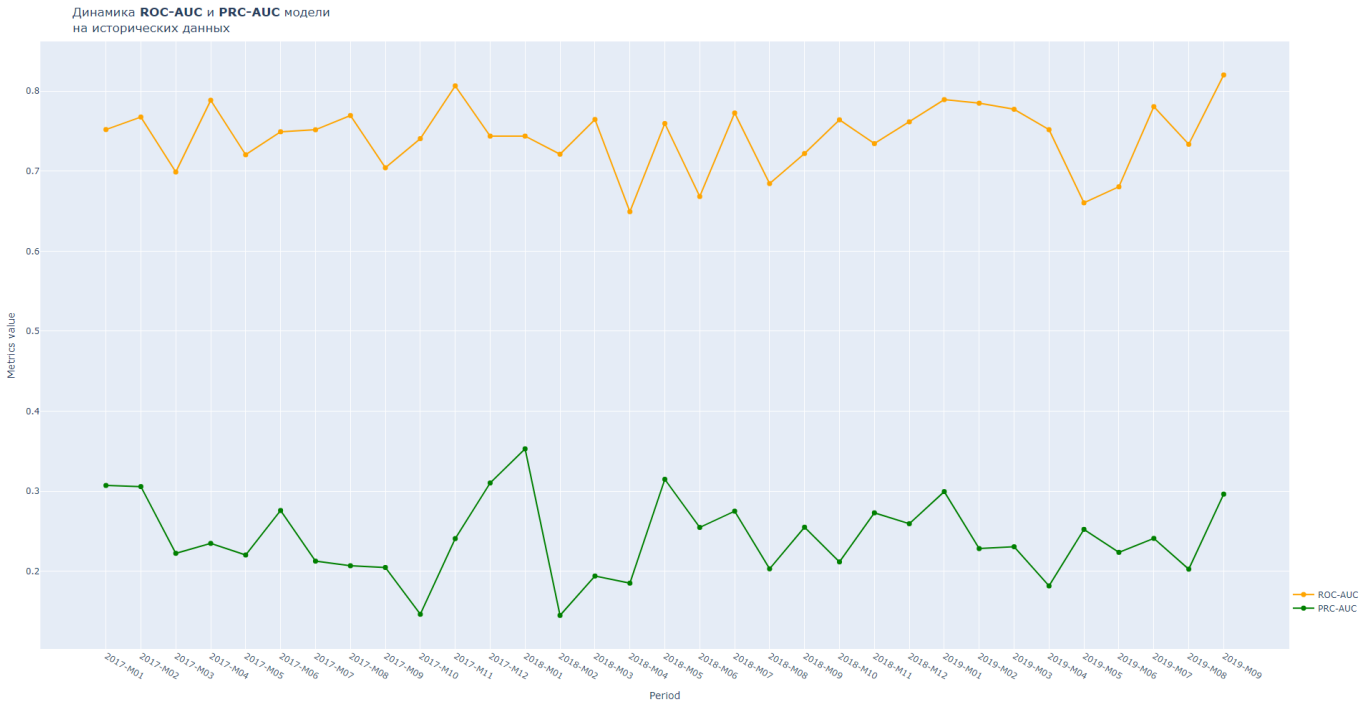
Output (long):

Массив графиков:
xaxis : период
yaxis: значения ROC AUC и PRC AUC, рассчитанные на наблюдениях за соответствующий период

Output (short):

Отсутствует

Output example (picture):



cvc_2_7_Gini_model

- **Техническое название**: cvc_2_7_Gini_model
- **Описание**: Gini Index (%) for Model. Индекс Джини (%) по модели
- **Теги**: core, classification, risk, scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`
- **Примечания**: -

Логика исполнения

1. Расчет ROC AUC по target_field, score_field
2. [Gini index] = 2 * [ROC AUC] - 1

Индекс Джини позволяет оценить качество предсказательной способности модели относительно случайного классификатора.
Чем выше индекс Джини, тем лучше модель справляется с разделением классов.
Джини от 0 до 1 (до 100%) - модель лучше случайного классификатора
Джини меньше 0 - модель хуже случайного классификатора
Пример выставления signal bounds: Gini ≤ 40(%) - красный,
40(%) < Gini ≤ 60(%) - желтый,
Gini > 60(%) - зеленый

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Число:
значение индекса Джини (в %),
Светофор

Output example (picture): не применимо.

cvc_2_8_KS_Test_Predicts

- **Техническое название**: cvc_2_8_KS_Test_Predicts
- **Описание**: Kolmogorov-Smirnov Test. Тест Колмогорова-Смирнова.
  Сравнивает распределения score для "хороших"/"плохих" клиентов
- **Теги**: core, classification, scalar
- **requirements**: `typing`, `pandas`, `numpy`, `scipy.stats`
- **Примечания**: -

Логика исполнения

Сравниваются распределения score для "хороших"(target==0) и "плохих"(target==1) клиентов.
H_0: распределения одинаковы.
Желательным является наличие статистически значимых различий между группами (это означ, что модель хорошо разделяет классы) => чем меньше p-value тем лучше
Реализация:
Двухвыборочный критерий Колмогорова-Смирнова ks_2samp из scipy.stats
(проверка гипотезы о принадлежности значений двух независимых выборок к одному и тому же закону распределения)

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

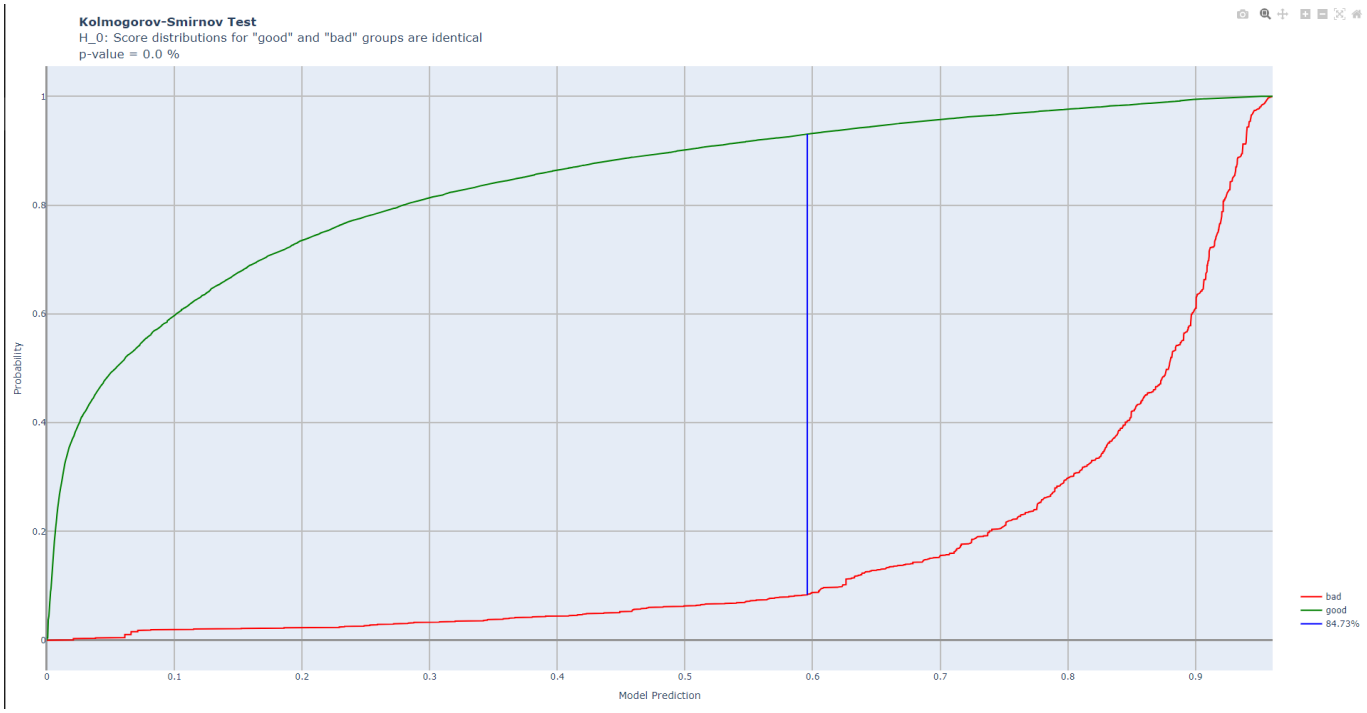
Output (long):

Массив графиков

Output (short):

Числовые значения:
1) Статистики К-С, % (скаляр)
2) p-value теста К-С, % (на графике)
светофор на Статистики К-С, %

Output example (picture):



cvc_2_9_Barometers_by_bins

- **Техническое название**: cvc_2_9_Barometers_by_bins
- **Описание**: Table of barometers by bins. Таблица показателей по бакетам
  Показатели по бакетам: точность, полнота, лифт, max, min и средний скор-балл, и др
- **Теги**: nan
- **requirements**: `typing`, `pandas`, `numpy`, `sklearn.metrics`
- **Примечания**: -

Логика исполнения

df разбивается на заданное число (nbins) бакетов по квантилям score.
Для каждого бакета (бина) выводятся:
- номер бина

- число записей в бине
- min score в бине
- max score в бине
- средний score в бине
- к-во записей с target==1 в бине
- cumsum (target) по агрегированной таблице
- cumsum (числа записей в бине) по агрегированной таблице
- recall (полнота)
- precision (точность)
- lift

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
nbins: число бакетов (int value; по умолчанию 20)

Результаты

Движок отрисовки графика: plotly.js

Output (long):

Таблица
11 столбцов (по числу рассчитываемых показателей),
число строк = числу бакетов

Output (short):

nan

Output example (picture):

Значения показателей по бакетам с шагом 7 %

Номер бакета	Всего записей	Min score	Max score	Средний score	Записей с target=1	Cumsum записей с target=1	Cumsum числа записей	Полнота (recall)	Точность (precision)	Lift
15	800	0.1889	0.7166	0.2751	225	225	800	1	0.2812	3.6562
14	800	0.139	0.1888	0.16	140	365	1600	0.9857	0.1767	2.2971
13	800	0.1125	0.139	0.1256	99	464	2400	0	0	1
12	799	0.0946	0.1125	0.1029	85	549	3199	0	0	1
11	793	0.0821	0.0946	0.088	59	608	3992	0	0	1
10	807	0.0705	0.082	0.0759	56	664	4799	0	0	1
9	801	0.0625	0.0705	0.0666	45	709	5600	0	0	1
8	799	0.0545	0.0625	0.0583	43	752	6399	0	0	1
7	800	0.0483	0.0545	0.0512	40	792	7199	0	0	1
6	784	0.0422	0.0483	0.0452	40	832	7983	0	0	1
5	816	0.0367	0.0422	0.0395	34	866	8799	0	0	1
4	800	0.0314	0.0367	0.0341	21	887	9599	0	0	1
3	800	0.0263	0.0314	0.0289	17	904	10399	0	0	1
2	799	0.0209	0.0263	0.0236	12	916	11198	0	0	1
1	801	0.0072	0.0209	0.0167	7	923	11999	0	0	1

Risk validation package

КАЧЕСТВО ДАННЫХ

r_1_1_PSI_field

- **Техническое название**: r_1_1_PSI_field

- **Описание**: PSI Features. Population stability index (PSI) for features
- **Теги**: risk
- **requirements**: `typing`, `pandas`, `numpy`, `plotly.graph_objects`

Примечание:

-

Логика исполнения

PSI - мера удаленности двух распределений
Для каждого из выбранных факторов определяется степень различия распределений этого фактора на выборках для обучения и тестирования
В цикле для каждого field из fields_to_test:
1. Если признак категориальный, то перейти к шагу 2. Если непрерывный, то нужно провести автоматический биннинг (10-20 бакетов).
2. Для каждой категории отдельно на train и test рассчитываем % наблюдений, принадлежащий этой категории.
3. temp_i = (%test - %train) * ln(%test / %train)
4. PSI = сумма temp_i по всем категориям
Возможный диапазон значений PSI: (0; +inf).
Чем ниже PSI, тем меньше отличаются распределения признака на train и test.
=> Чем ниже PSI для фактора, тем лучше (тем стабильнее) этот фактор
Пример выставления signal bounds: PSI < 0.1 - зеленый,
0.1 < PSI < 0.25 - желтый,
PSI > 0.25 - красный

Входные параметры

df_train: датасет с данными для обучения (dataframe)
df_test: датасет с данными для теста (dataframe)
field_columns: массив названий атрибутов, по которым считаем PSI
(multi-column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора
(! названия столбцов из field_columns должны совпадать в df_train и df_test)

Результаты

Движок отрисовки графика: plotly.js

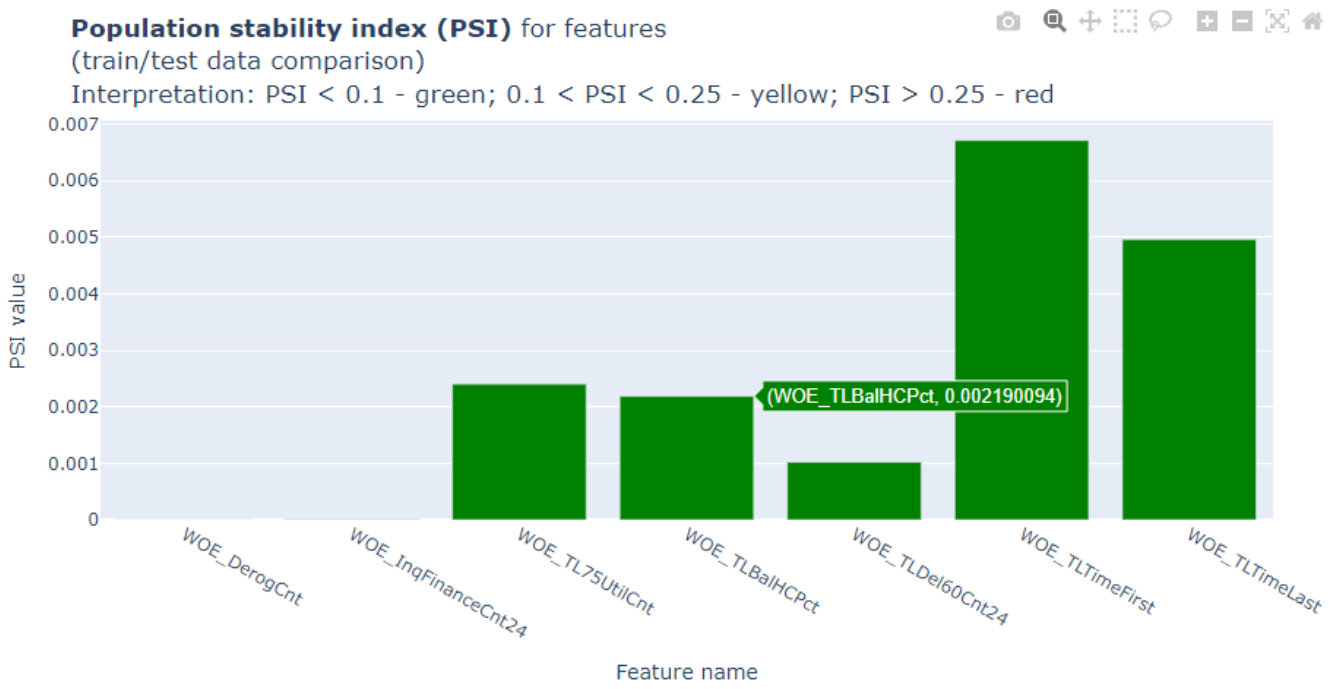
Output (long):

Барчарт
xaxis: Столбцы соответствуют признакам, для которых производился расчет
yaxis : Высота столбца - значение PSI для признака
Светофор: столбцы окрашиваются в цвет светофора для соответствующего признака

Output (short):

Отсутствует

Output example (picture):



r_1_2_Default_rate_dynamic

- **Техническое название**: r_1_2_Default_rate_dynamic
- **Описание**: Default Rate Dynamic. Уровень дефолта на исторических данных:
  - доля (%) наблюдений с дефолтом по периодам,
  - общее к-во наблюдений по периодам.
- **Теги**: risk
- **requirements**: `typing`, `pandas`, `plotly.graph_objects`

Примечание:

-

Логика исполнения

Группируем данные с гранулярностью period.
Для каждой группы (периода) отрисовываем на комбинированном графике:
- количество наблюдений
- % наблюдений с дефолтом =
(к-во наблюдений с [target=1] за период) / (к-во наблюдений за период) * 100%
Тест обычно используется для первичной валидации модели.
(Например, для оценки соотношения классов в выборках,
поиска периодичности во времени,
отсечения слишком старых и неактуальных данных)

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
report_dt_column: название столбца с датой (column)
period: гранулярность данных
(dropdown - одно из значений типа str: 'month', 'quarter', 'year';
по умолчанию -'quarter')

Результаты

Движок отрисовки графика: plotly.js

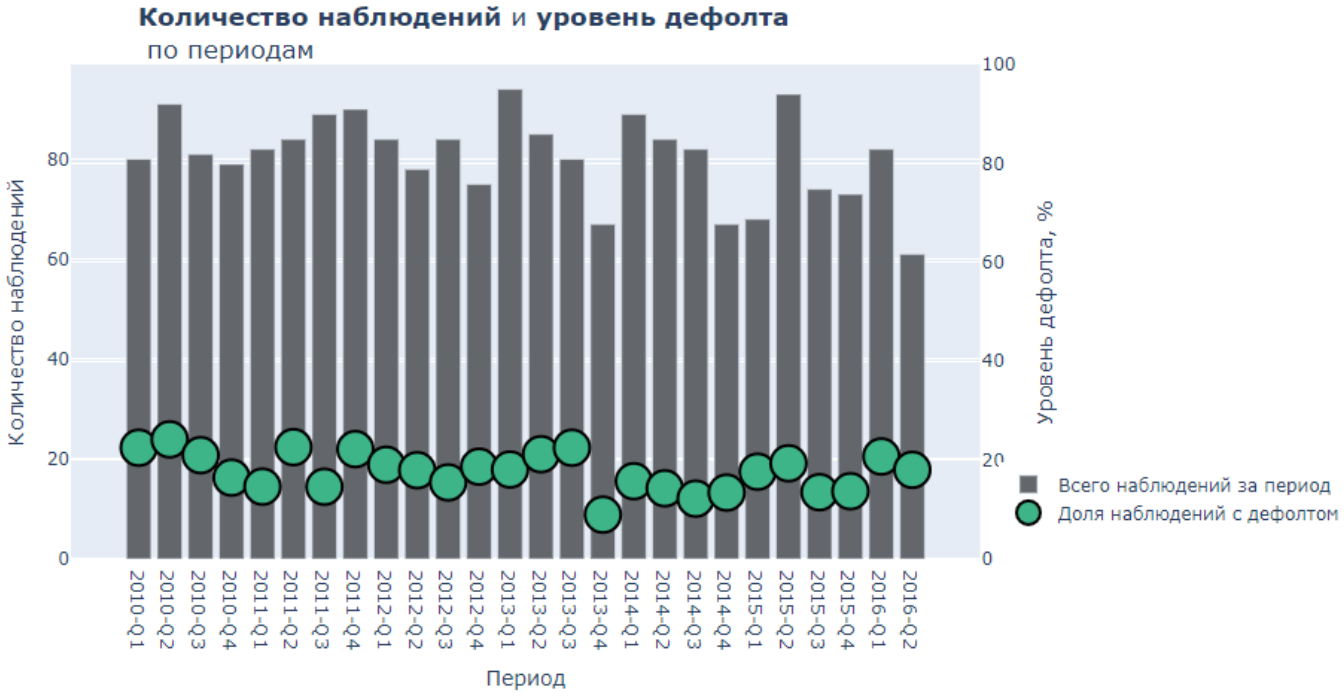
Output (long):

Барчарт
xaxis: Столбцы соответствуют периодам
yaxis :
Высота столбца (шкала слева) - число наблюдений в датасете, относящихся к выбранному периоду
Точка (шкала справа) - доля (%) наблюдений с дефолтом в выбранный период

Output (short):

Отсутствует

Output example (picture):



РАНЖИРУЮЩАЯ СПОСОБНОСТЬ

cvc_2_7_Gini_model (r_2_1_Gini_model)

- **Техническое название**: cvc_2_7_Gini_model (r_2_1_Gini_model)
- **Описание**: Gini Index (%) for Model. Индекс Джини (%) по модели
- **Теги**: -
- **requirements**: -

Примечание:

-

Логика исполнения

См. раздел "Core package/Метрики качества классификации", пункт 2.7

Входные параметры

-

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Отсутствует

Output example (picture): не применимо.

cd_4_2_Gini_features (r_2_2_Gini_features)

- **Техническое название**: cd_4_2_Gini_features (r_2_2_Gini_features)
- **Описание**: Gini Index (%) for Features. Индекс Джини (%) в разрезе отдельных факторов
- **Теги**: -
- **requirements**: -

Примечание:

-

Логика исполнения

См. раздел "Core package/Анализ данных", пункт 4.2

Входные параметры

-

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Отсутствует

Output example (picture): не применимо.

r_2_3_Gini_bootstrap_model

- **Техническое название**: r_2_3_Gini_bootstrap_model
- **Описание**: Bootstrapped Gini Index (%) for Model. Индекс Джини (%) по модели на bootstrap-подвыборках
  Интервальная версия точечной оценки Джини (см. 2.1)
  Используется при наличии дисбаланса классов в выборке.
- **Теги**: risk,
  scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`, `joblib`, `numpy`

Примечание:

-

Логика исполнения

1. Собираем выборку из bootstrap_n значений Gini, рассчитанных на bootstrap-подвыборках
2. Выбираем из полученной выборки перцентиль perc

При несбалансированной выборке это позволяет избежать смещения точечной оценки.
По умолчанию берется 2.5 перцентиль распределения Gini.
Чем больше значение Bootstrapped Gini, тем лучше предсказательная способность модели.
Пример выставления signal bounds: Bootstrapped Gini ≤ 30(%) - красный,
30(%) < Bootstrapped Gini ≤ 45(%) - желтый,
Bootstrapped Gini > 45(%) - зеленый

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
bootstrap_n: количество bootstrap-подвыборок (int value; по умолчанию 1000)
perc: перцентиль, по которому проводим итоговую оценку (float value; по умолчанию 2.5)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Число:
значение индекса Джини (в %)
(2.5 перцентиль распределения Джини на бутстраппированных подвыборках из исходного df),
светофор

Output example (picture): не применимо.

r_2_4_Gini_bootstrap_field

- **Техническое название**: r_2_4_Gini_bootstrap_field
- **Описание**: Bootstrapped Gini Index (%) for Features. Индекс Джини (%) в разрезе отдельных факторов на bootstrap-подвыборках
  Интервальная версия оценки Джини для факторов (см. 2.2)
  Используется при наличии дисбаланса классов в выборке.
- **Теги**: risk
- **requirements**: `typing`, `pandas`, `sklearn.metrics`, `joblib`, `numpy`

Примечание:

-

Логика исполнения

В цикле для каждого field из fields:
1. Собираем выборку из bootstrap_n значений Gini, рассчитанных на bootstrap-подвыборках
2. Выбираем из полученной выборки перцентиль perc

При несбалансированной выборке это позволяет избежать смещения точечной оценки.
По умолчанию берется 2.5 перцентиль распределения Gini.
Если в признаке (field) или target присутствует пропущенное значение, такая строка исключается из рассмотрения. Для разных признаков исключаются из рассмотрения разные строки.
Чем больше значение Bootstrapped Gini для фактора, тем лучше ранжирующая способность этого фактора.

Пример выставления signal bounds: Bootstrapped Gini ≤ 3(%) - красный,
3(%) < Bootstrapped Gini ≤ 5(%) - желтый,
Bootstrapped Gini > 5(%) - зеленый

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
field_columns: массив названий столбцов, по которым считаем тест (multi-column)
bootstrap_n: количество bootstrap-подвыборок (int value; по умолчанию 1000)
perc: перцентиль, по которому ставим итоговую оценку (float value; по умолчанию 2.5)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

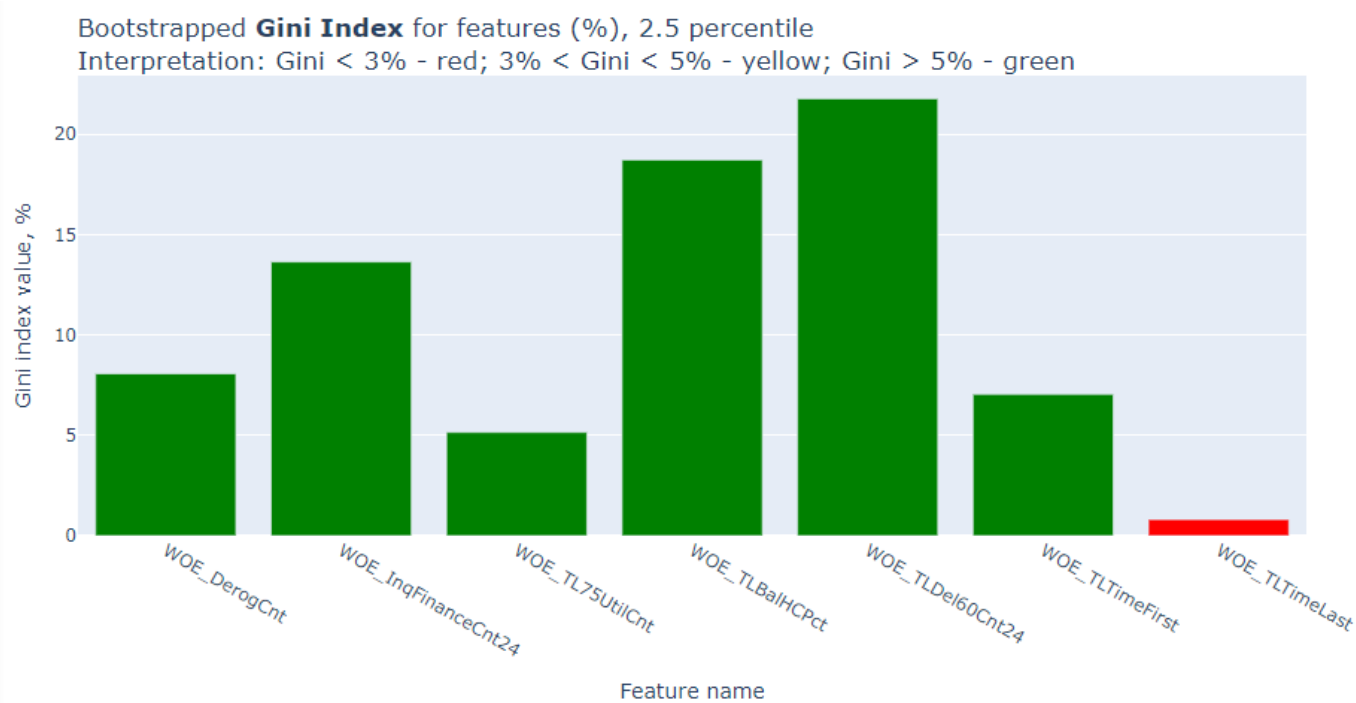
Output (long):

Барчарт:
xaxis : названия столбцов, для которых производился расчет
yaxis : значения коэффициента Джини (в %)
(2.5 перцентиль распределения Джини для фактора на бутстраппированных подвыборках из
Светофор: столбцы окрашиваются в цвет светофора для соответствующего признака

Output (short):

Отсутствует

Output example (picture):



r_2_5_KS_on_scale

- **Техническое название**: r_2_5_KS_on_scale

- **Описание**: KS-test on scale. Тест Колмогорова-Смирнова.
  Проверяет насколько отличаются 2 распределения -
  показывает насколько хорошо score модели отделяет хороших клиентов от плохих в разрезе рейтинговой шкалы.
- **Теги**: risk,
  scalar
- **requirements**: `typing`, `pandas`, `numpy`

Примечание:

-

Логика исполнения

1. Собираем 2 кумулятивных распределения:
   1. Good - доля клиентов с [target_field = 0] с группировкой по scale_field
   2. Bad- доля клиентов с [target_field = 1] с группировкой по scale_field
   2. Считаем значение статистики max(\Good-Bad) по всем разрядам шкалы. Чем больше, чем лучше.
   Альтернатива cvc_2_8_KS_Test_Predicts с использованием scale шкалы
   Обычно разряды рейтинговой шкалы присваиваются на основании score модели.
   Формирование scale: по предсказанию модели (score) каждому из наблюдений присваивается разряд рейтинговой шкалы => получаем категориальную переменную
   Интерпретация графика: для хорошей модели кривые bad и good д.б. удалены друг от друга.
   H_0 теста К-С: выборки взяты из одного распределения.
   Нам нужно, чтобы bad и good максимально отличались =>
   Чем больше значение статистики К-С, тем лучше.
   Пример выставления signal bounds: КС ≤ 30(%) - красный,
   30(%) < КС ≤ 40(%) - желтый,
   КС > 40(%) - зеленый

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
scale_column: название столбца с присвоенным разрядом рейтинговой шкалы (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

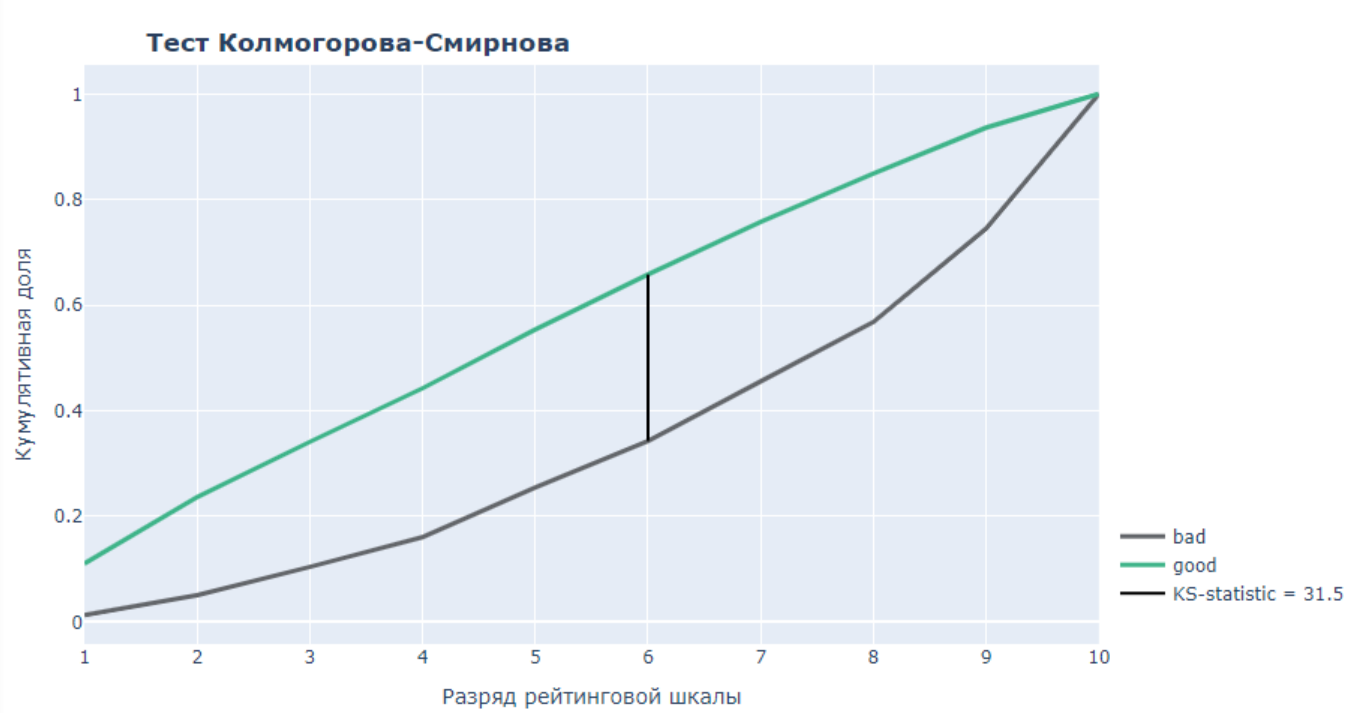
Output (long):

Массив графиков:
xaxis : разряды рейтинговой шкалы
yaxis : кумулятивная вероятность,
Значение статистики К-С (в %) - в легенде

Output (short):

Число: Значение статистики К-С (в %)
Светофор

Output example (picture):



r_2_6_IV_model

- **Техническое название**: r_2_6_IV_model
- **Описание**: Information Value для модели
  Показывает степень отличия распределений predict для хороших и плохих клиентов
- **Теги**: risk,
  scalar
- **requirements**: `typing`, `pandas`, `numpy`, `scipy.stats.stats`, `sklearn`, +, Преобразование WOE, созданное отдельным классом в том же файле метрики

Примечание:

-

Логика исполнения

1. Провести автоматический биннинг score_field (10-20 бакетов).
2. Для каждой категории рассчитываем:
1. %good - процент наблюдений с [target=0]
2. %bad- процент наблюдений с [target=1]
3. temp_i = (%good- %bad) * ln(%good/ %bad)
4. IV = сумма temp_i по всем категориям

Формула аналогична PSI, но если PSI сравнивает распр-е переменной на разных временных срезах, то IV сравнивает распределение score для хороших и плохих клиентов (на "хороших" и "плохох" разбиваем по знач-ю target)
Чем сильнее отлич-ся эти распр-я, тем лучше =>
чем больше IV, тем лучше.
Пример выставления signal bounds: IV ≤ 10(%) - красный,
10(%) < IV ≤ 30(%) - желтый,
IV > 30(%) - зеленый

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)

predict_column: название столбца со скором (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Число:
значение Information Value для модели (в %),
светофор

Output example (picture): не применимо.

r_2_7_IV_field

- **Техническое название**: r_2_7_IV_field
- **Описание**: Information value в разрезе отдельных факторов
  Показывает степень отличия распределений фактора для хороших и плохих клиентов
- **Теги**: risk
- **requirements**: `typing`, `pandas`, `numpy`, `scipy.stats.stats`, `sklearn`, +, Преобразование WOE, созданное отдельным классом в том же файле метрики

Примечание:

-

Логика исполнения

В цикле для каждого field из fields:
1. Если field категориальный, то перейти к п.2, иначе провести автоматический биннинг (10-20 бакетов).
2. Для каждой категории рассчитываем:
1. %good - процент наблюдений с [target=0]
2. %bad- процент наблюдений с [target=1]
3. temp_i = (%good- %bad) * ln(%good/ %bad)
4. IV = сумма temp_i по всем категориям
Чем больше IV для признака, тем лучше (тем информативнее этот признак).
Тест используется для отбора признаков модели. (См. статью о WOE и IV)
Пример выставления signal bounds: IV ≤ 10(%) - красный,
10(%) < IV ≤ 30(%)- желтый,
IV > 30(%) - зеленый

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
field_columns: массив названий столбцов, по которым считаем тест
(multi-column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

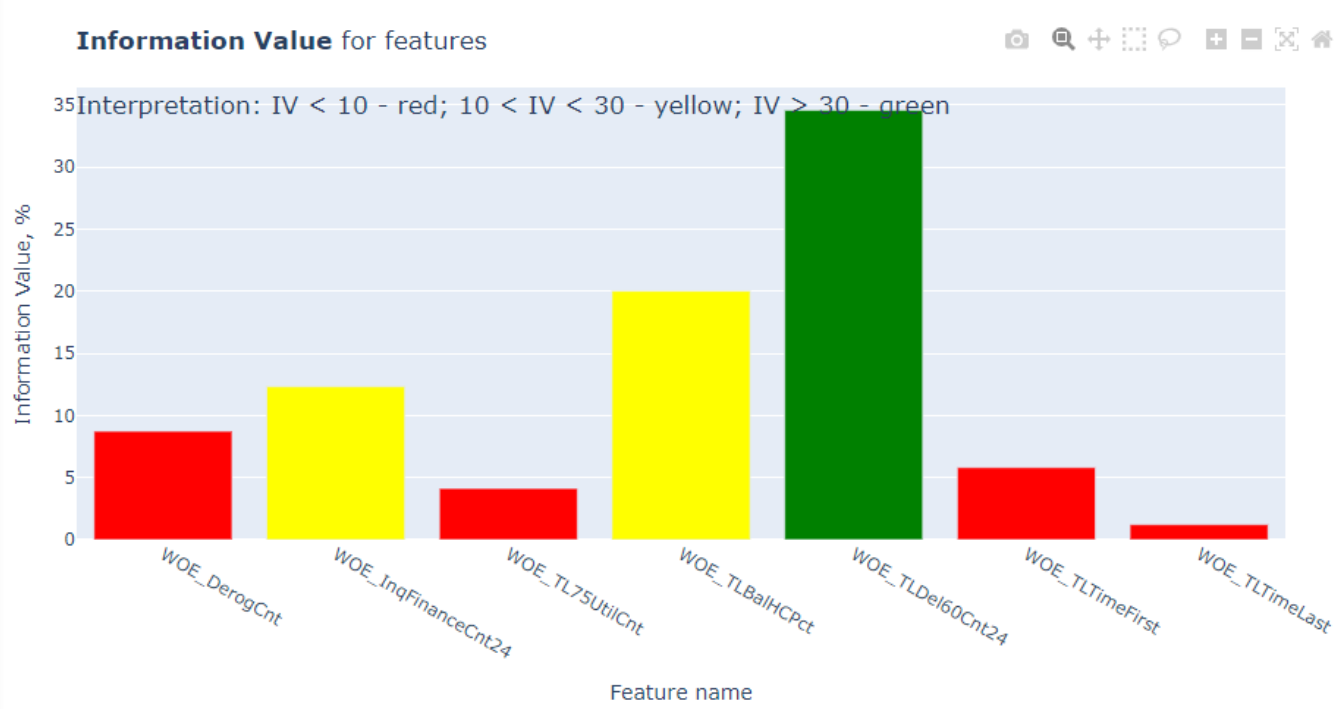
Output (long):

Барчарт:
xaxis : названия столбцов, для которых производился расчет
yaxis : значения Information Value (в %)
Светофор: столбцы окрашиваются в цвет светофора для соответствующего признака

Output (short):

Отсутствует

Output example (picture):



r_2_8_HL_test

- **Техническое название**: r_2_8_HL_test
- **Описание**: Hosmer–Lemeshow Test. Тест Хосмера-Лемешова (хи-квадрат)
  Проверяет схожесть распределений среднего фактического и среднего прогнозного уровня дефолта по бакетам
- **Теги**: risk,
  scalar
- **requirements**: `typing`, `pandas`, `scipy.stats`

Примечание:

-

Логика исполнения

1. Для i-го разряда шкалы рассчитываются параметры:
   1. N_i - количество наблюдений в бакете
   2. PD_i - среднее значение score_field по бакету (прогнозный уровень дефолта)
   3. DR_i - среднее значение target_field по бакету (фактический уровень дефолта)
   2. Рассчитывается статистика Хосмера-Лемешова по формуле

$$T_n = \sum_{i=0}^{n} \frac{(N_i PD_i - DR_i)^2}{N_i PD_i (1 - PD_i)},$$

   3. p-value определяется по распределению хи-квадрат (scipy.stats.chi2)

H_0: распределения по группам одинаковы.
Мы стремимся к тому, чтобы распределения по бакетам для target и score были как можно более схожи => Чем выше p-value, тем лучше
Пример выставления signal bounds:
p-value ≤ 1(%) - красный,
1(%) < p-value ≤ 5(%)- желтый,
p-value > 5(%) - зеленый
Используется также для калибровки модели.

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
scale_column: название столбца с присвоенным разрядом рейтинговой шкалы (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Число:
p-value теста Хосмера-Лемешова (в %),
светофор

Output example (picture): не применимо.

r_2_9_MW_test

- **Техническое название**: r_2_9_MW_test
- **Описание**: Mann-Whitney Test. Тест Манна-Уитни (левосторонний)
  Проверка схожести распределений score и target по бакетам
  (ранговый метод)
- **Теги**: risk,
  scalar
- **requirements**: `typing`, `pandas`, `scipy.stats`

Примечание:

-

Логика исполнения

1. Разбиваем score на nbins бакетов по квантилям --> получим категориальную переменную с nbins значений.
2. Группируем наблюдения по новой категориальной переменной. В каждой группе рассчитываем mean(score) и mean(target) --> получим две выборки: средние score по бакетам и средние target по бакетам
3. С помощью непараметрического критерия Манна-Уитни (scipy.stats.mannwhitneyu) проверяем гипотезу о принадлежности этих выборок к одной и той же генеральной совокупности.
4. Выводим p-value.

Мы стремимся чтобы распределения по бакетам для score и target были как можно более схожи => Чем выше p-value, тем лучше
Используется для: калибровки модели, проверки стабильности.
Пример выставления signal bounds:
p-value ≤ 1(%) - красный,
1(%) < p-value ≤ 5(%)- желтый,
p-value > 5(%) - зеленый

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
nbins: число бакетов, на которое score разбивается по квантилям
(int value; по умолчанию 50)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Число:
p-value теста Манна-Уитни (в %),
светофор

Output example (picture): не применимо.

СПЕЦИФИКАЦИЯ (ТОЛЬКО ДЛЯ ЛИНЕЙНЫХ МОДЕЛИ)

r_3_1_Monotony_field

- **Техническое название**: r_3_1_Monotony_field
- **Описание**: Monotony Field. Монотонность фактора.
  Доля дефолтов и количество наблюдений по категориям фактора на train/test
- **Теги**: risk
- **requirements**: `typing`, `pandas`

Примечание:

-

Логика исполнения

Разбиваем наблюдения на группы в соответствии со значениями категориальной переменной field. Число групп = числу уникальных значений field. Для категориального field отрисовываем комбинированный график:

1. Барчарт :
% наблюдений в каждой категории field на train и test =
= (наблюдений в выбранной категории) / (всего наблюдений в выборке) * 100
2. Линия:
% дефолтов в каждой категории field на train и test =
= (наблюдений с target==1 в выбранной категории) / (всего наблюдений в выбранной категории) * 100
Тест применяется если в модели использовались биннинги или WOE (Weight of Evidence).
(Для упорядоченных WOE доля дефолтов должна строго возрастать)

Входные параметры

df_train: датасет с данными для обучения (dataframe)
df_test: датасет с данными для теста (dataframe)
target_column: название столбца с таргетом (column)
cat_field_column: название столбца с категориальной переменной по значениям которой разбиваем наблюдения на группы (column)
(! названия столбцов с target_column и cat_field_column должны совпадать в df_train и df_test)

Результаты

Движок отрисовки графика: plotly.js

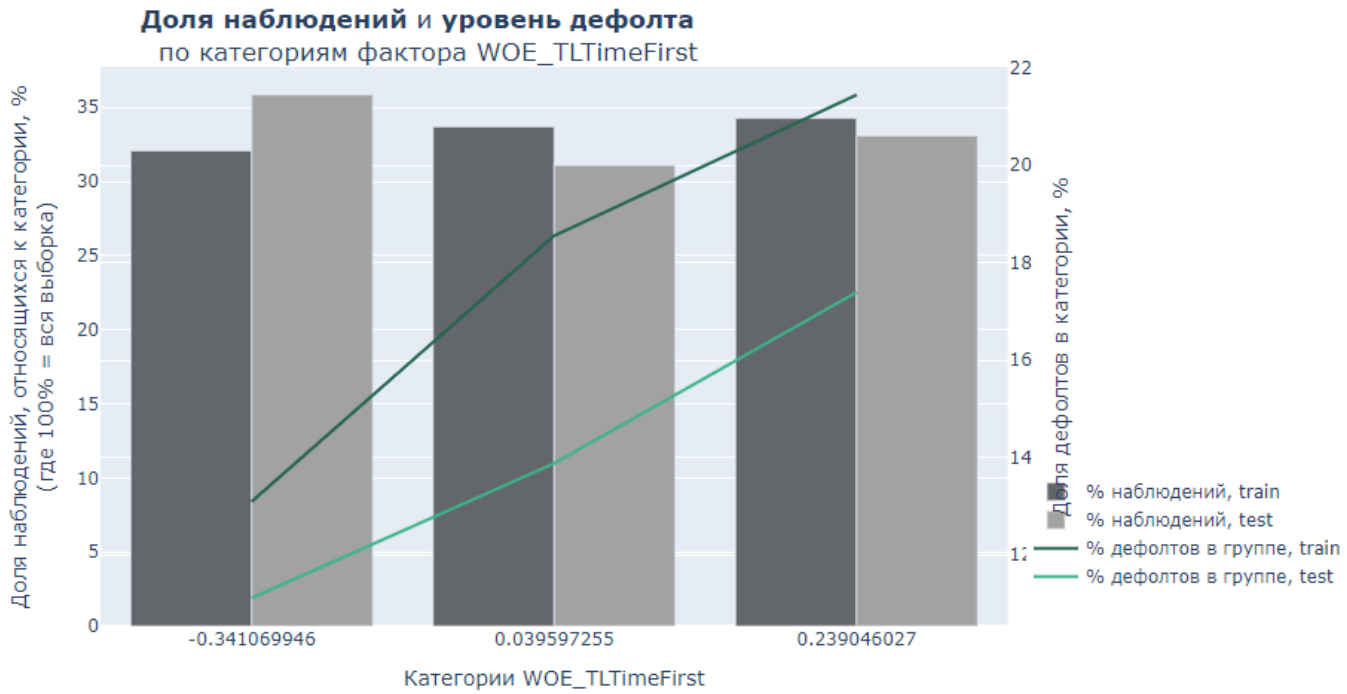
Output (long):

Массив графиков
Барчарт:
xaxis: каждая пара столбцов соответствует одному значению категориальной переменной field на train/test
yaxis (шкала слева) : высота столбца = доля всех наблюдений, относящихся к выбранной категории
Линии:
yaxis (шкала справа): доля дефолтов в выбранной категории

Output (short):

Отсутствует

Output example (picture):



cd_4_4_VIF (r_3_2_VIF)

- **Техническое название**: cd_4_4_VIF (r_3_2_VIF)

- **Описание**: Variance Inflation Factor. Коэффициент инфляции дисперсии (используется для обнаружения мультиколлинеарности)
- **Теги**: -
- **requirements**: -

Примечание:

-

Логика исполнения

См. раздел "Core package/Анализ данных", пункт 4.4

Входные параметры

-

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Отсутствует

Output example (picture): не применимо.

r_3_3_Spearman_Correlation

- **Техническое название**: r_3_3_Spearman_Correlation
- **Описание**: Spearman Correlations. Матрица попарных ранговых корреляций Спирмена и максимальное значение корреляции (%)
- **Теги**: risk, scalar
- **requirements**: `typing`, `pandas`

Примечание:

-

Логика исполнения

Отрисовываем heatmap с матрицей парных корреляций всех признаков из fields_to_test
Пропущенные поля исключаем из рассмотрения.
Корреляция Спирмена - непараметрическая мера статистической зависимости между двумя переменными. Она оценивает, насколько хорошо можно описать зависимость между переменными монотонной функцией.
При отсутствии повторяющихся значений коэфф Спирмена +1 или -1 свидетельствует о том, что одна переменная является строго монотонной функцией другой (зависимость не обязательно линейная).
Коэфф корреляции Спирмена менее чувствителен к выбросам, чем коэфф корреляции Пирсона.
Пример выставления signal bounds:
max(Corr) ≥ 50- красный,
40 ≤ max(Corr) < 50 - желтый,
max(Corr) < 40 - зеленый

Входные параметры

df: датасет с данными для исследования (dataframe)
field_columns: массив названий столбцов, по которым считаем тест (multi-column)

threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

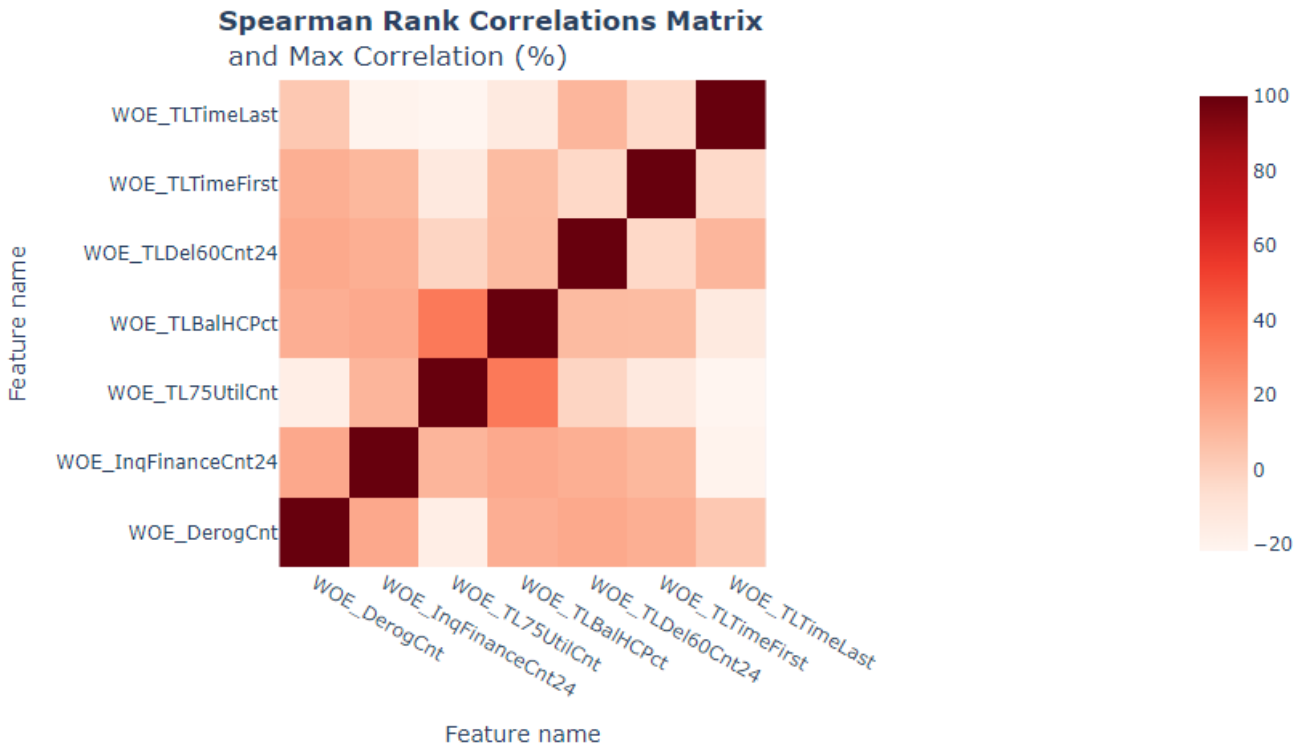
Output (long):

Heatmap:
Матрица попарных корреляций выбранных столбцов

Output (short):

Число:
максимальное из значений парной корреляции для выбранных столбцов, в %
Светофор

Output example (picture):



СТАБИЛЬНОСТЬ

r_4_1_Gini_diff_model

- **Техническое название**: r_4_1_Gini_diff_model
- **Описание**: Abs Gini Difference (%) train/test. Абсолютное значение изменения Gini Index модели (в %) на train/test
  Показывает, насколько отличается предсказательная способность модели на разных выборках
- **Теги**: risk,
  scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`

Примечание:

-

Логика исполнения

abs([Gini index_test] - [Gini index_train])
Чем меньше значение разницы Gini. тем лучше.
Если Gini на тесте значительно меньше, чем на train, то произошло переобучение.
Пример выставления signal bounds:
\Δ Gini\ < 5% - зеленый,
5% ≤ \Δ Gini\ < 10% - желтый,
\Δ Gini\ ≥ 10% - красный

Входные параметры

df_train: датасет с данными для обучения (dataframe)
df_test: датасет с данными для теста (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора
(! названия столбцов с target_column и predict_column должны совпадать в df_train и df_test)

Результаты

Движок отрисовки графика: plotly.js

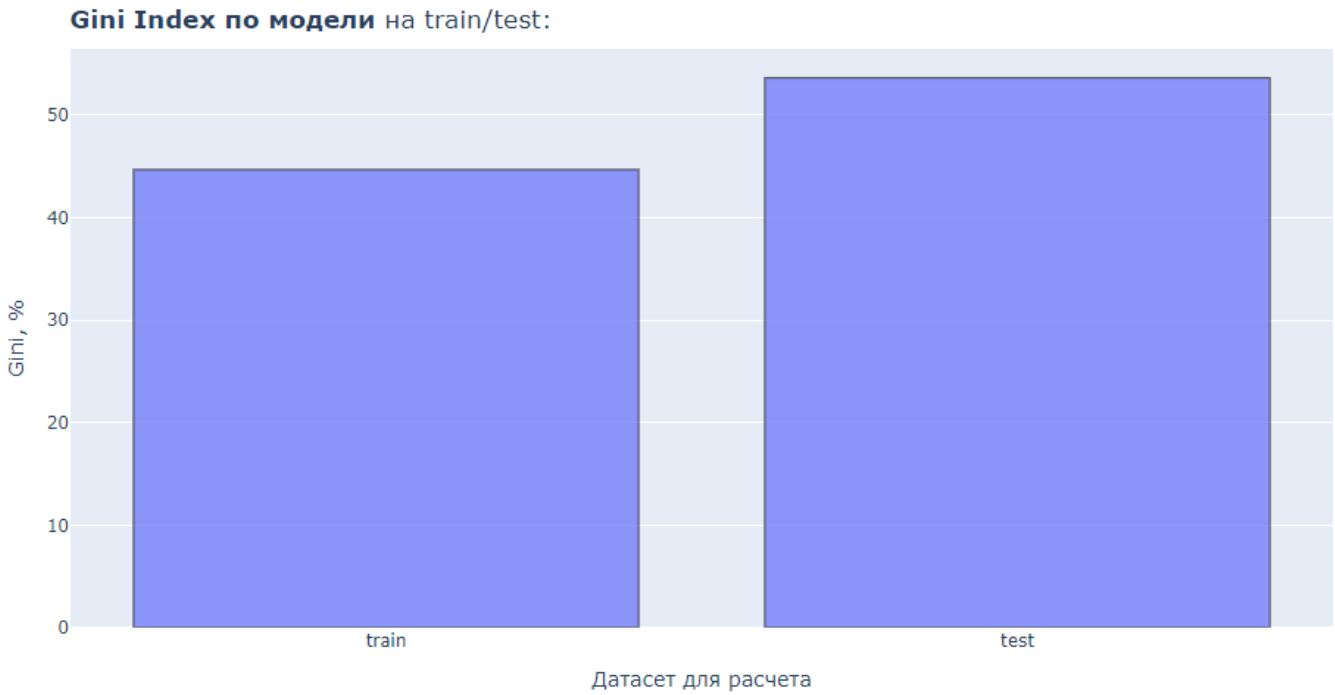
Output (long):

Барчарт:
2 столбца - соответствуют выборкам train и test.
Высота столбца - значение Gini по модели на указанной выборке.

Output (short):

Число:
разница Gini на train и test в %
(выводится в составе заголовка)

Output example (picture):



r_4_2_Gini_reltv_diff_model

- **Техническое название**: r_4_2_Gini_reltv_diff_model

- **Описание**: Gini Ind model - train/test reltv diff (%). Относительное изменение Gini index модели (в %) на train/test
  Показывает насколько улучшилась/ухудшилась предсказательная способность модели на test по отношению к предсказательной способности на train
- **Теги**: risk,
  scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`

Примечание:

-

Логика исполнения

([Gini index_test] - [Gini index_train]) / [Gini index_train]
Чем меньше модуль показателя, тем лучше
Пример выставления signal bounds:
\Δ Gini\ < 10% - зеленый,
10% ≤ \Δ Gini\ < 20% - желтый,
\Δ Gini\ ≥ 20% - красный

Входные параметры

df_train: датасет с данными для обучения (dataframe)
df_test: датасет с данными для теста (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора
(! названия столбцов с target_column и predict_column должны совпадать в df_train и df_test)

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Число: относительное изменение Gini по модели, в %,
Светофор

Output example (picture): не применимо.

r_4_3_Gini_diff_features

- **Техническое название**: r_4_3_Gini_diff_features
- **Описание**: Gini Ind features - train/test diff (%). Относительное изменение Gini index на train/test в разрезе отдельных факторов.
  Показывает, насколько улучшилась/ухудшилась информативность каждого из факторов на test по отношению к информативности этого фактора на train
- **Теги**: risk
- **requirements**: `typing`, `pandas`, `sklearn.metrics`

Примечание:

-

Логика исполнения

В цикле для каждого категориального field из fields считаем:
abs([Gini index_test] - [Gini index_train])

Если в признаке (field) или target присутствует пропущенное значение, такая строка исключается из рассмотрения.
Для разных признаков исключаются из рассмотрения разные строки.
Чем меньше модули показателей, тем лучше
Пример выставления signal bounds на относительное изменение индекса Джини по факторам между двумя соседними срезами данных:
\Δ Gini\ < 10% - зеленый,
10% ≤ \Δ Gini\ < 15% - желтый,
\Δ Gini\ ≥ 15% - красный

Входные параметры

df_train: датасет с данными для обучения (dataframe)
df_test: датасет с данными для теста (dataframe)
target_column: название столбца с таргетом (column)
field_columns: массив названий столбцов, по которым считаем тест
(multi-column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора
(! названия столбцов с target_column и field_columns должны совпадать в df_train и df_test)

Результаты

Движок отрисовки графика: plotly.js

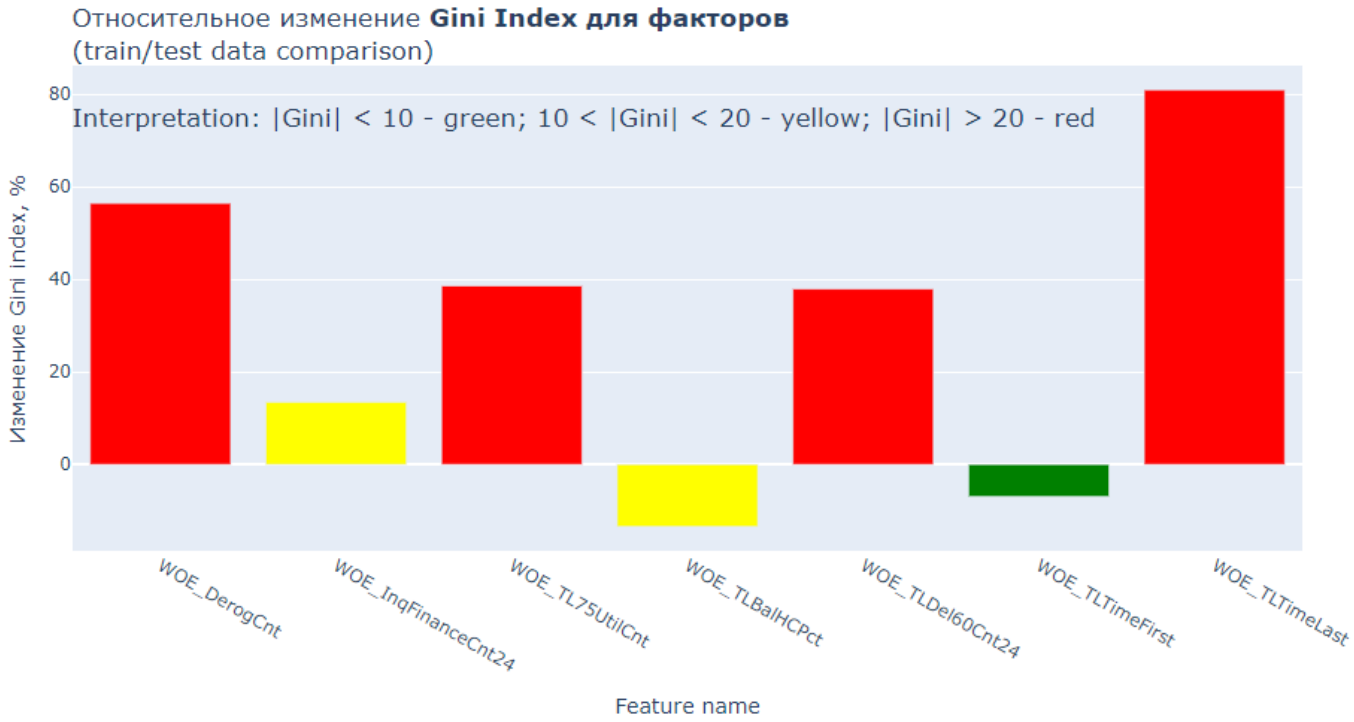
Output (long):

Барчарт:
Столбцы соответствуют признакам
Высота столбца - отличие Gini для соответствующего признака на train и test.
Светофор: столбцы окрашиваются в цвет светофора для соответствующего признака

Output (short):

Отсутствует

Output example (picture):



r_4_4_Gini_dynamic_model

- **Техническое название**: r_4_4_Gini_dynamic_model
- **Описание**: Gini Index (model) Dynamic. Динамика индекса Джини модели.
  (Bar-chart со значениями Джини по модели в разные периоды времени)
- **Теги**: risk,
  scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`, `numpy`

Примечание:

-

Логика исполнения

Наблюдения из df разбиваются на группы по значениям столбца с датой (report_dt). В одну группу попадают все наблюдения за period (месяцы, квартал или год).
Для наблюдений каждой группы рассчитывается индекс Джини по модели.
Чем больше Джини, тем лучше модель.
Джини меньше 0 означает, что результаты модели хуже случайного угадывания.
В динамике: чем стабильнее Джини, тем лучше.
Пример выставления signal bounds на абсолютное изменение индекса Джини по модели между двумя соседними срезами данных:
\Δ Gini\ < 5% - зеленый,
5% ≤ \Δ Gini\ < 10% - желтый,
\Δ Gini\ ≥ 10% - красный

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
report_dt_column: название столбца с датой (column)
period: гранулярность данных
(dropdown - одно из значений типа str: 'month', 'quarter', 'year'; по умолчанию -'year')
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

Output (long):

Барчарт:
Столбцы соответствуют периодам
Высота столбца - значение индекса Gini по модели, рассчитанного на наблюдениях за соответствующий период
Точное значение Gini отображается при наведении мышки на соответствующий столбец

Output (short):

Число:
максимальное изменение Gini

Output example (picture):

r_4_5_PSI_model

- **Техническое название**: r_4_5_PSI_model
- **Описание**: PSI Model. Population stability index (PSI) по модели
- **Теги**: risk,
  scalar
- **requirements**: `typing`, `pandas`, `numpy`

Примечание:

Реализовано автоматическое разбиение score на заданное число бакетов. В большинстве моделей допустимо автоматическое разбиение на заданное число бакетов, без дополнительной шкалы.
Альтернативой может выступать разделение по мастер-шкале в отдельном столбце. Мастер-шкала составляется банком. В рисковых моделях обычно используют разбиение через мастер-шкалу.

Логика исполнения

Проводится проверка различия распределений предсказаний модели (score) на выборках для обучения и тестирования
1. Для каждого бакета (разбиение по значениям score) отдельно на train и test рассчитываем % наблюдений, принадлежащий этому бакету
2. temp_i = (%test - %train) * ln(%test / %train)
3. PSI = сумма temp_i по всем категориям
Корреляция с качеством модели - отрицательная. (Чем ниже PSI, тем лучше модель)
Пример выставления signal bounds: PSI < 0.1 - зеленый,
0.1 < PSI < 0.25 - желтый,
PSI > 0.25 - красный

Входные параметры

df_train: датасет с данными для обучения (dataframe)
df_test: датасет с данными для теста (dataframe)
predict_column: название столбца со скором (column)
nbuckets: число бакетов, на которое разбивается score
(int value; по умолчанию 30)
threshold_yellow: желтая граница светофора

threshold_red: красная граница светофора
(! названия столбца со predict_column должно совпадать в df_train и df_test)

Результаты

Движок отрисовки графика: -

Output (long):

Отсутствует

Output (short):

Число:
значение PSI по модели между train и test
Светофор

Output example (picture): не применимо.

r_4_6_Lift_dynamic

- **Техническое название**: r_4_6_Lift_dynamic
- **Описание**: Lift Dynamic. Lift в динамике
- **Теги**: risk
- **requirements**: `typing`, `pandas`, `sklearn.metrics`, `numpy`

Примечание:

-

Логика исполнения

Значения Lift по периодам
Группируем данные с гранулярностью period. Для каждой группы отрисовываем на барчарте значения lift за каждый период:
lift = [доля наблюдений с predict_field=1] / [доля наблюдений с target_field=1]
lift = TPR/PR
В динамике: чем стабильнее Lift, тем лучше

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
report_dt_column: название столбца с датой (column)
period: гранулярность данных
(dropdown - одно из значений типа str: 'month', 'quarter', 'year';
по умолчанию -'year')

Результаты

Движок отрисовки графика: plotly.js

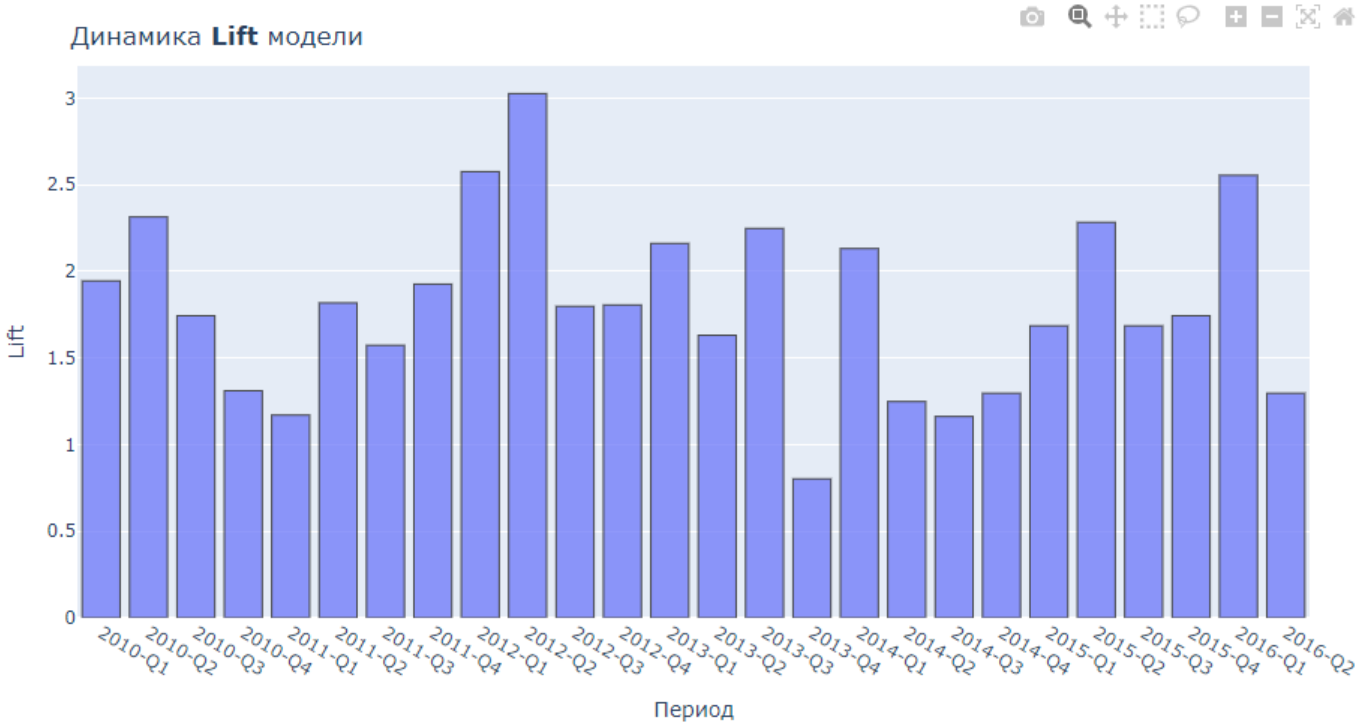
Output (long):

Барчарт:
Столбцы соответствуют периодам
Высота столбца - значение Lift, рассчитанное на наблюдениях за соответствующий период
Точное значение Lift отображается при наведении мышки на соответствующий столбец

Output (short):

Отсутствует

Output example (picture):



r_4_7_reltv_PR_Curve

- **Техническое название**: r_4_7_reltv_PR_Curve
- **Описание**: PR Curves train/test. Графики Precision-Reccall Curves (кривые точности-полноты) для train/test и относительное изменение PRC AUC
- **Теги**: risk, scalar
- **requirements**: `typing`, `pandas`, `sklearn.metrics`

Примечание:

-

Логика исполнения

([PR AUC_test] - [PR AUC_train]) / [PR AUC_train]
Характеризует качество модели классификации.
Чем больше (ближе к 1) значение PRC AUC, тем лучше модель.
PRC полезна в задачах , где объектов класса 1 мало, но их необходимо выявлять.
В динамике: чем стабильнее PRC AUC, тем лучше.

Входные параметры

df_train: датасет с данными для обучения (dataframe)
df_test: датасет с данными для теста (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора
(! названия столбцов с target_column и predict_column должны совпадать в df_train и df_test)

Результаты

Движок отрисовки графика: plotly.js

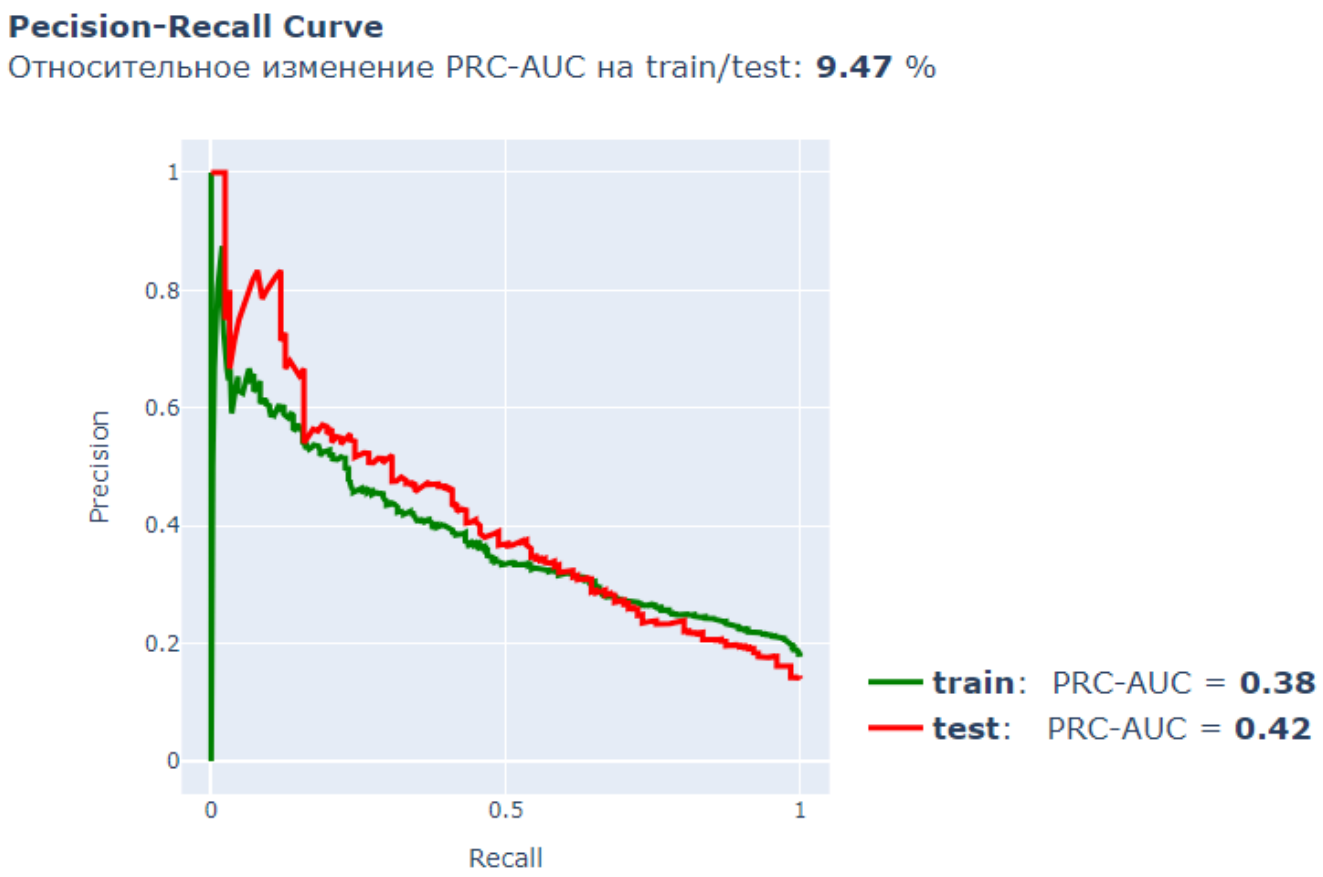
Output (long):

PR curves для обучающей и тестовой выборок

Output (short):

Число:
относительное изменение PR AUC на train/test

Output example (picture):



КОНЦЕНТРАЦИЯ

r_5_1_HHI

- **Техническое название**: r_5_1_HHI
- **Описание**: Herfindahl-Hirschman Index (HHI). Индекс Херфиндаля-Хиршмана
  и барчарт с распределением наблюдений по разрядам рейтинговой шкалы
- **Теги**: risk,
  scalar
- **requirements**: `typing`, `pandas`

Примечание: -

Логика исполнения

Индекс Херфиндаля-Хиршмана описывает равномерность распределения по разрядам рейтинговой шкалы
1. Для i-го разряда рейтинговой шкалы рассчитываем N_i - количество наблюдений

$$HI = \sum_i \left(\frac{N_i}{N}\right)^2,$$

2. , где N - общее число наблюдений в датасете
Чем меньше индекс XX, тем лучше
(тем равномернее распределение по группам)

Пример выставления signal bounds:
HHI < 20% - зеленый,
20% ≤ HHI < 30% - желтый,
HHI ≥ 30% - красный

Входные параметры

df: датасет с данными для исследования (dataframe)
scale_column: название столбца с присвоенным разрядом рейтинговой шкалы (column)
threshold_yellow: желтая граница светофора
threshold_red: красная граница светофора

Результаты

Движок отрисовки графика: plotly.js

Output (long):

Барчарт:
xaxis: Разряды рейтинговой шкалы
yaxis:
Количество наблюдений, относящихся к соответствующему разряду

Output (short):

Число:
значение индекса Херфиндаля-Хиршмана для выбранного столбца, в %
Светофор

Output example (picture):

r_5_2_Unique_clients

- **Техническое название**: r_5_2_Unique_clients

- **Описание**: Unique Clients by Bins. Гистограммы распределения по разрядам рейтинговой шкалы для:
  - количества значений,
  - количества уникальных значений
  выбранного столбца (например, столбца с id клиентов)
- **Теги**: risk
- **requirements**: `typing`, `pandas`

Примечание:

-

Логика исполнения

Отрисовываем комбинированный барчарт с:
- общим числом наблюдений в каждом бакете рейтинговой шкалы
- числом уникальных значений в каждом бакете рейтинговой шкалы
Пример применения метрики:
исследование распределения уникальных клиентов по рейтинговой шкале (в переменную field подаем название столбца с id клиентов. Получим: всего клиентов по группам, уникальных клиентов по группам)

Входные параметры

df: датасет с данными для исследования (dataframe)
scale_column: название столбца с присвоенным разрядом рейтинговой шкалы (column)
field_column: название столбца, для которого проверяем распределение значений по разрядам (column)

Результаты

Движок отрисовки графика: plotly.js

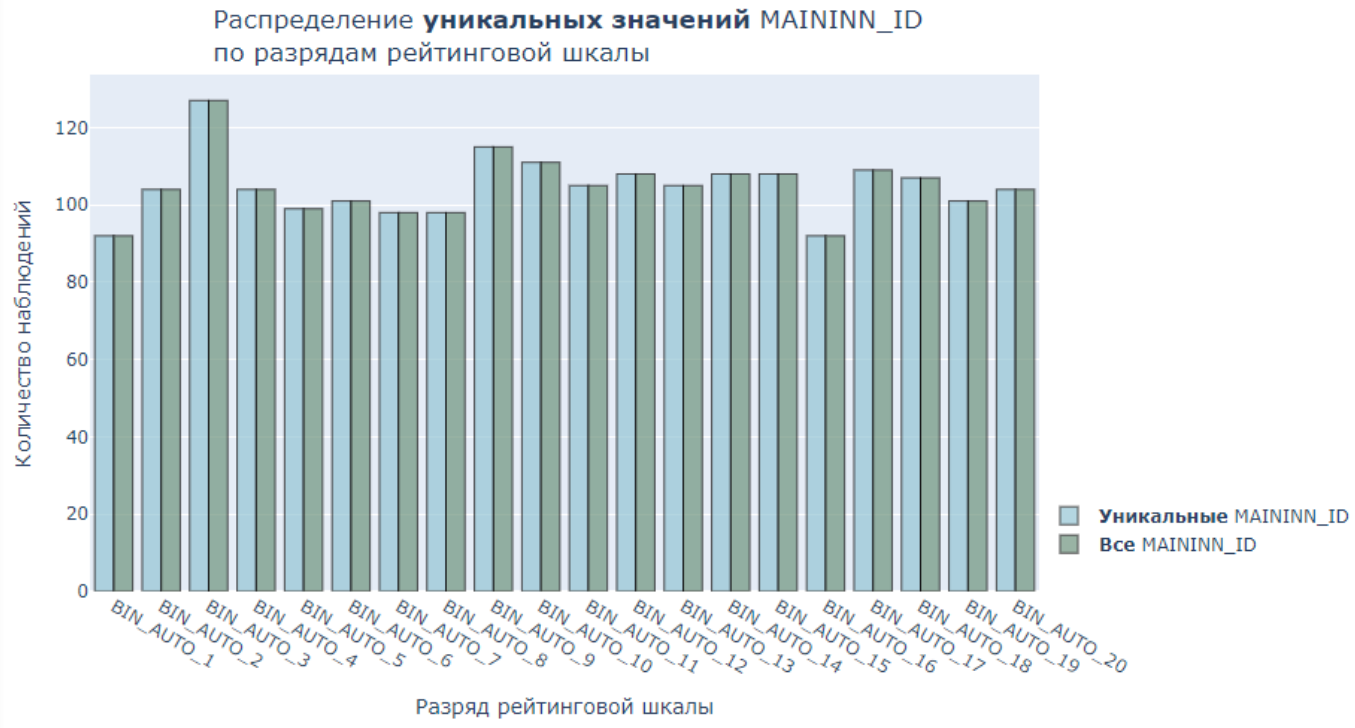
Output (long):

Барчарт:
xaxis: Пара столбцов соответствует разряду рейтинговой шкалы.
yaxis:
Высота столбцов: левого - число уникальных значений в группе, правого - общее число наблюдений в группе

Output (short):

Отсутствует

Output example (picture):



r_5_3_Derivative_Score_Distribution

- **Техническое название**: r_5_3_Derivative_Score_Distribution
- **Описание**: Derivative Score Distribution. Распределение производных оценок
- **Теги**: risk
- **requirements**: `typing`, `pandas`, `numpy`

Примечание:

-

Логика исполнения

DSD обычно описывает распределение производных оценок, что может помочь в анализе чувствительности модели к изменениям входных данных. Отрисовываем линейный график, представляющий собой график зависимости производной функции доли наблюдений от скора, начиная со значения, соответствующего перцентилю perc_threshold

Входные параметры

df: датасет с данными для исследования (dataframe)
predict_column: название столбца со скором (column)
perc_threshold: граница перцентиля для отсечения значений predict_column (float)

Результаты

Движок отрисовки графика: plotly.js

Output (long):

Линия:
xaxis: Значения предсказаний модели
yaxis:
Прирост процента наблюдений значений предсказаний модели в датасете

Output (short):

Отсутствует

Output example (picture):

КАЛИБРОВКА

r_6_1_Model_default_rate

- **Техническое название**: r_6_1_Model_default_rate
- **Описание**: Model Default Rate. Модельный и фактический уровень дефолта по бакетам рейтинговой шкалы
- **Теги**: risk
- **requirements**: `typing`, `pandas`

Примечание:

-

Логика исполнения

Прогнозный и фактический default rate находим как средние значения score и target по разрядам рейтинговой шкалы.
Отрисовываем линейные графики:
1. Среднее значение target_field по i-ому разряду рейтинговой шкалы
2. Среднее значение score_field по i-ому разряду рейтинговой шкалы
Чем ближе др. к др. графики (прогнозный и реальный default rate), тем лучше

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
scale_column: название столбца с присвоенным разрядом рейтинговой шкалы (column)

Результаты

Движок отрисовки графика: plotly.js

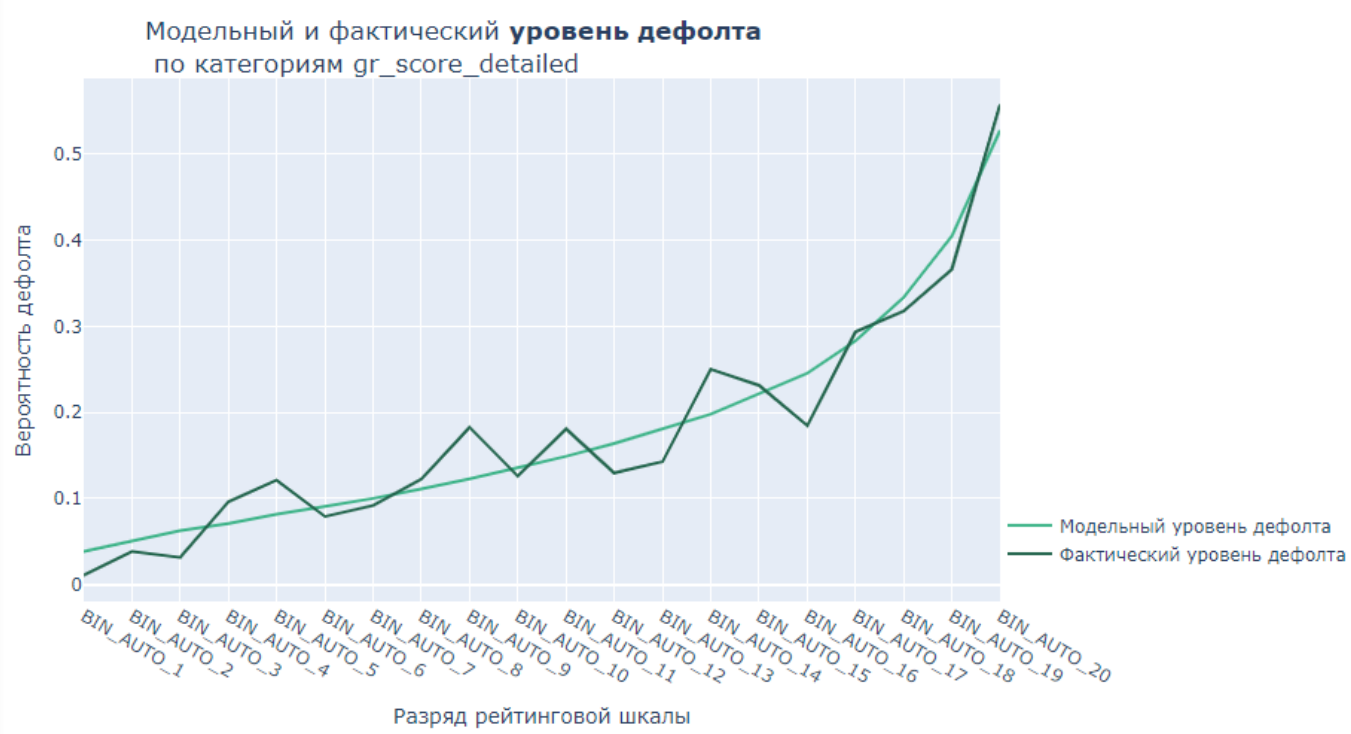
Output (long):

Массив графиков:
xaxis: разряд рейтинговой шкалы
yaxis: вероятность дефолта
Графики соответствуют модельному и фактическому уровню дефолта

Output (short):

Отсутствует

Output example (picture):



r_6_2_Binomial_test

- **Техническое название**: r_6_2_Binomial_test
- **Описание**: Binomial Test. Биномиальный тест
  Для каждой группы рейтинговой шкалы проверяется попадание фактического значения дефолта (среднего target по группе) в доверительный интервал для среднего значения предсказания (score) по группе
- **Теги**: risk
- **requirements**: `typing`, `pandas`, `scipy.stats`, `numpy`

Примечание:

-

Логика исполнения

1. Для i-го бакета рейтинговой шкалы на основе распределения score_field строим доверительный интервал по формуле:

$$PD_i - \varphi^{-1}(CI)\sqrt{PD_i * (1 - PD_i)/n_i};\ PD_i + \varphi^{-1}(CI)\sqrt{PD_i * (1 - PD_i)/n_i},$$ где

$\varphi^{-1}(CI)\sqrt{PD_i * (1 - PD_i)/n_i}$ – критическое значение уровня дефолтов, аппроксимированное с помощью нормального распределения,

φ^{-1} – обратная функция нормального распределения,

CI – уровень доверия,

PD_i - средний score_field по бакету

2. Для успешного прохождения теста средний target_field должен попадать в доверительный интервал по для каждого бакета рейтинговой шкалы

Входные параметры

df: датасет с данными для исследования (dataframe)
target_column: название столбца с таргетом (column)
predict_column: название столбца со скором (column)
scale_column: название столбца с присвоенным разрядом рейтинговой шкалы (column)
confidence_level: уровень доверия (float value; по умолчанию 0.99)

Результаты

Движок отрисовки графика: plotly.js

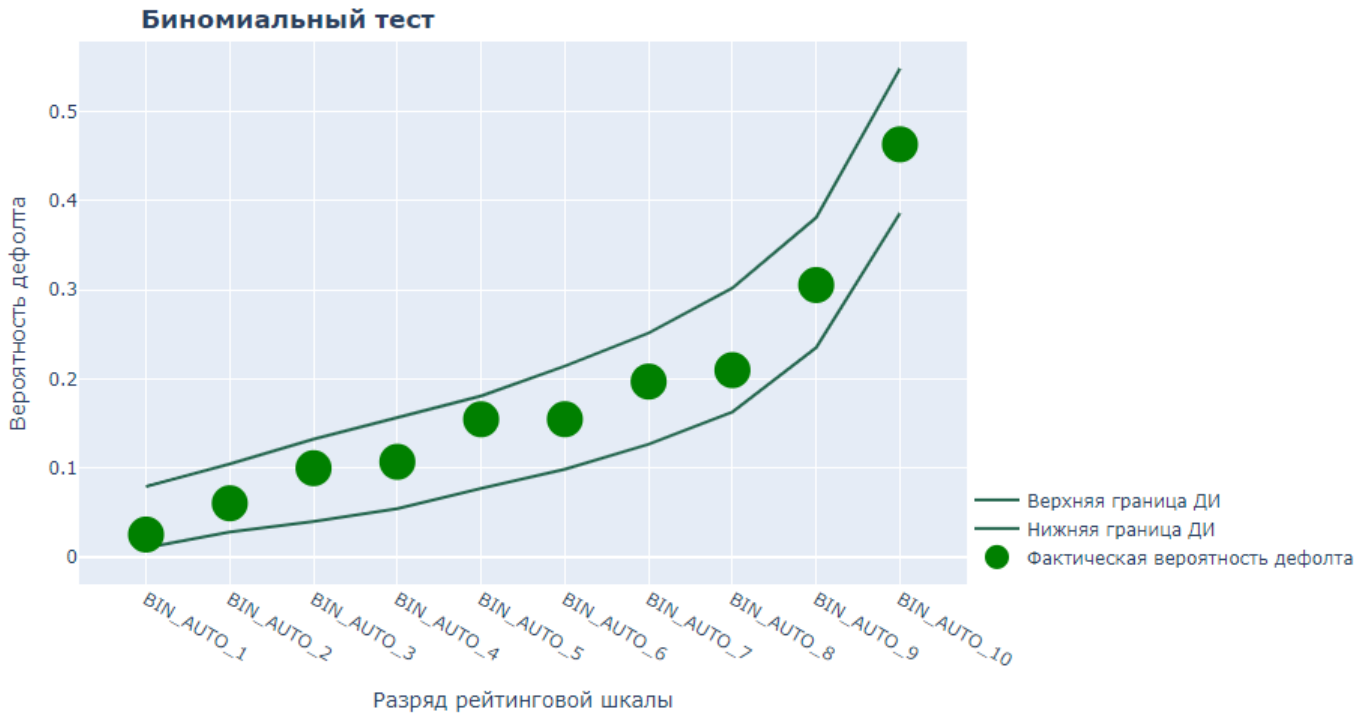
Output (long):

Массив графиков
xaxis: разряд рейтинговой шкалы
yaxis: вероятность дефолта

Output (short):

Отсутствует

Output example (picture):



Data drift package

DD_1_ADWIN

- **Техническое название**: dd_1_ADWIN
- **Описание**: Расчет Data Drift методом ADWIN (Adaptive Windowing)
- **Теги**: data_drift
- **requirements**: `typing`, `pandas`, `river`
- **Примечания**: -

Логика исполнения

Алгоритм выстраивает данные по временному ряду. Далее алгоритм проходит по данным оконным методом и вычисляет средние значения на этих окнах. Если обнаруживается значимое различие между близлежащими окнами - регистрируется событие дрейфа данных
Более подробно о методе ADWIN

Входные параметры

df: датасет с данными для исследования (dataframe)
field_column: название столбца для расчета (column)

Результаты

Движок отрисовки графика: plotly.js

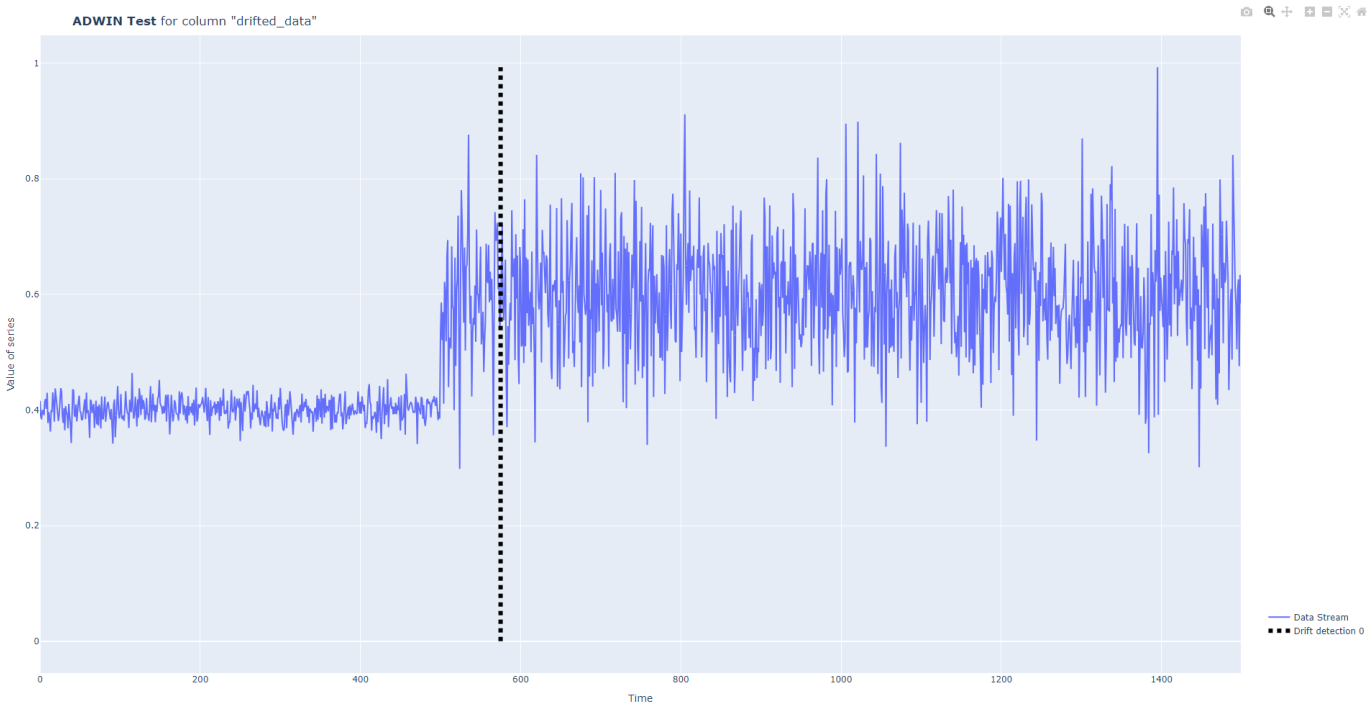
Output (long):

Массив графиков:
1. График значений столбца в виде линейного графика
2. Вертикальные границы, отмечающие события дрейфа данных, найденные алгоритмом

Output (short):

Отсутствует

Output example (picture):



DD_2_PAGEHINKLEYTEST

- **Техническое название**: dd_2_PageHinkleyTest
- **Описание**: Расчет Data Drift методом Page-Hinkley
- **Теги**: data_drift
- **requirements**: `typing`, `pandas`, `river`
- **Примечания**: -

Логика исполнения

Базово алгоритм похож на ADWIN (п. 1)
Более подробно о методе Page Hinkley Test

Входные параметры

df: датасет с данными для исследования (dataframe)
field_column: название столбца для расчета (column)

Результаты

Движок отрисовки графика: plotly.js

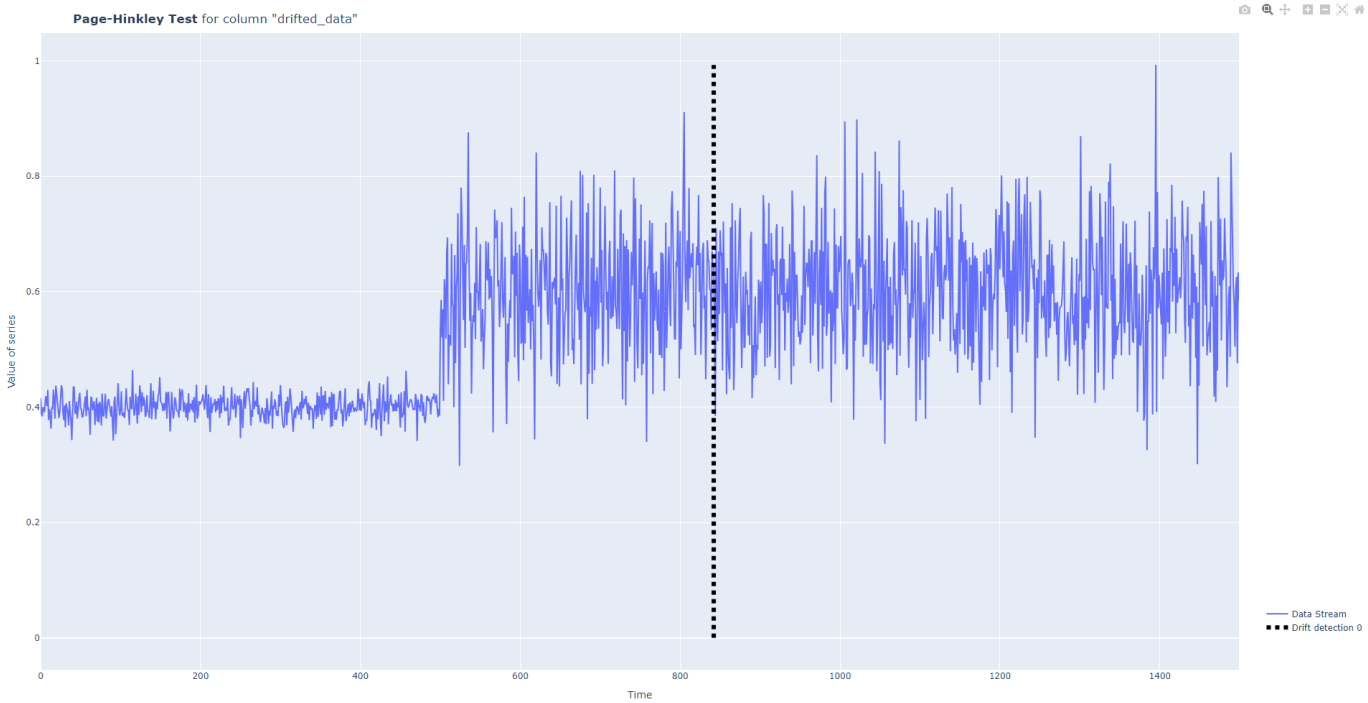
Output (long):

Массив графиков:
1. График значений столбца в виде линейного графика
2. Вертикальные границы, отмечающие события дрейфа данных, найденные алгоритмом

Output (short):

Отсутствует

Output example (picture):



2.3.2 Регистрация новой метрики

Список метрик из стандартного пакета доступны на этой странице

Подготовка файла метрики

Predicate поддерживает работу с метриками, которые написаны на языке Python.

Загрузка метрики или теста в систему осуществляется из файла с расширением `.py`.

В базовой версии приложения используются следующие версии языка и пакетов:

Развернуть перечень пакетов

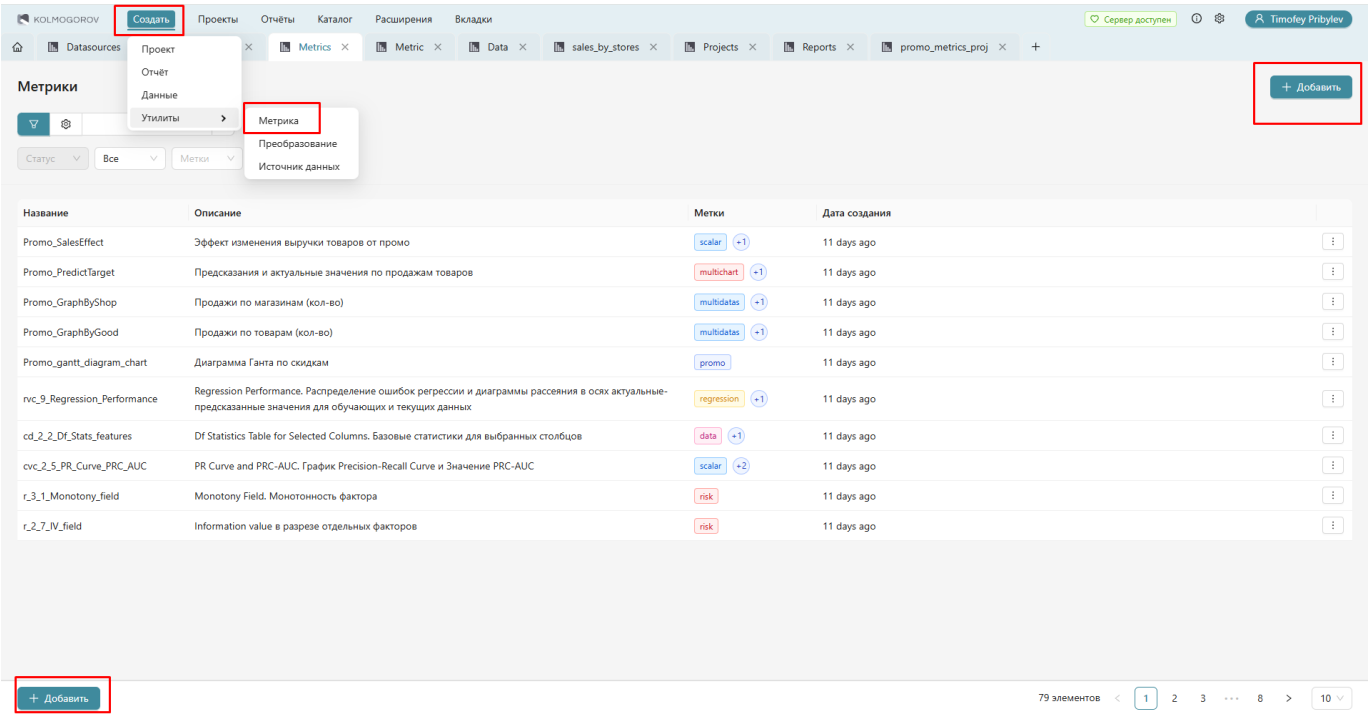
Python версии 3.11
pandas версии 2.2.0
nympy версии 1.26.4
plotly версии 5.20.0
matplotlib версии не ниже 3.9.2
scikit-learn версии не ниже 1.2.0
statsmodels версии не ниже 0.13.5
river версии не ниже 0.16.0

Использование других версий языка и пакетов и добавление новых пакетов возможно, но требует дополнительной настройки приложения. Свяжитесь с администраторами системы для проведения нужных работ.

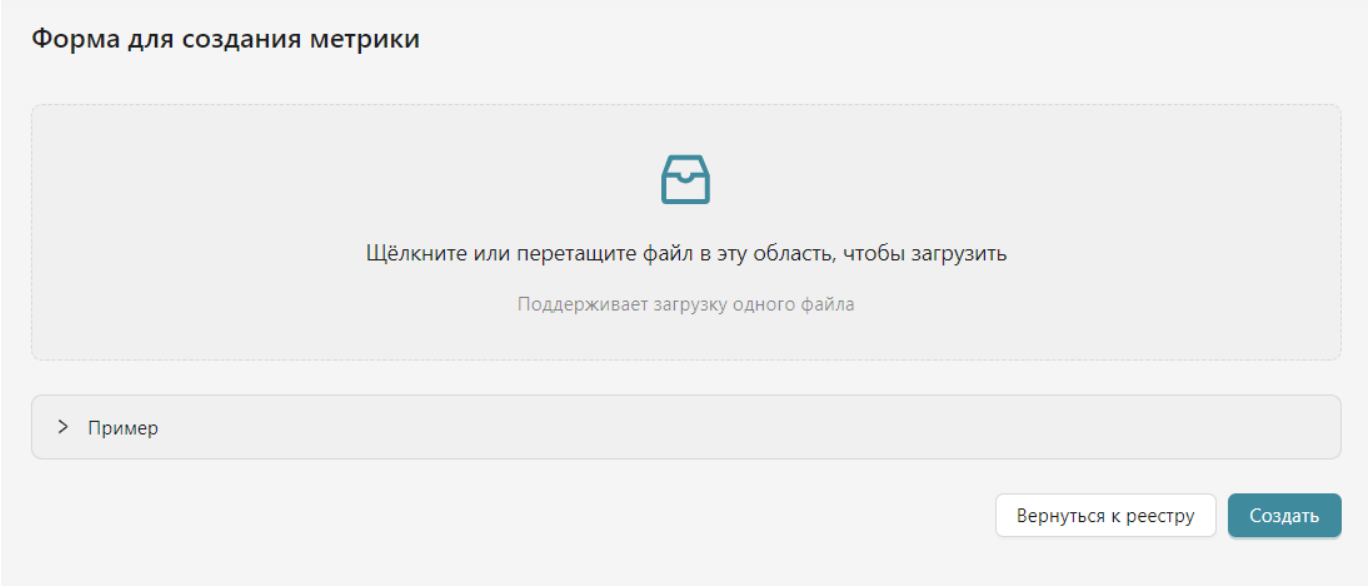
Подробнее о составлении кода метрики см. на странице "Правила написания метрик".

Регистрация метрики

Для загрузки новой метрики перейдите по пути *Панель управления* > *Каталог* > *Метрики* в основном меню приложения и на открывшейся странице каталога метрик и тестов нажмите кнопку "Добавить". Также добавление метрики доступно по пути *Панель управления* > *Создать* > *Утилиты* > *Метрика*.



Откроется форма загрузки новой метрики:



В открывшейся форме необходимо загрузить python-файл с кодом метрики и нажать кнопку "Создать".

После того как метрика создана, соответствующая ей запись появится в списке тестов, зарегистрированных в системе (*Каталог > Метрики*), и метрика станет доступна для использования в проектах мониторинга.

2.3.3 Правила написания метрик

Метрики Predicate - это класс Python с определенной структурой. Данный класс не содержит специфичных зависимостей, кроме `pandas`.

Примеры кода метрики доступны на отдельной странице.

На данной странице описаны:

1. Структура класса с метрикой
2. Рекомендации по написанию метрик

Структура класса с метрикой

У класса метрики есть набор обязательных полей и методов. При этом нет ограничений на использование дополнительных полей или методов. Мы даже посоветуем использовать дополнительные методы и поля в разделе "Рекомендации по написанию метрик".

Обязательные поля:

1. `__desc__`;
2. `__tags__`;
3. `is_scalar`;
4. `is_signal`;

Обязательные методы:

1. `__init__`;
2. `__call__`;
3. `save`;
4. `scalar` (если флаг `is_scalar = True`);
5. `signal` (если флаг `is_signal = True`).

В разделах ниже поля и методы описаны подробнее.

ПОЛЯ

Название класса становится названием метрики в системе

`__desc__` - понятное человеку описание метрики

`__tags__` - список тегов, присвоенных метрике в системе

`is_scalar` - флаг, что метрика имеет один скалярный результат. Если `is_scalar = True`, то метрика обязана содержать метод `scalar`

`is_signal` - флаг, что метрика имеет светофор. Если `is_signal = True`, то метрика обязана содержать метод `signal`

Таким образом, класс метрики "Тест Колмогорова-Смирнова", которая содержит и график, и скалярное значение, и светофор, должен начинаться следующим образом:

```
class r_2_5_KS_on_scale:
    __desc__ = "KS-test on scale. Тест Колмогорова-Смирнова"
    __tags__ = ["risk", "scalar"]

    is_scalar = True
    is_signal = True
```

МЕТОДЫ

`__init__`

Метод инициализации класса предназначен для ввода параметров метрики. Именно в этот метод будут передаваться все параметры метрики при вызове метрики в проекте Predicate или в рамках библиотеки.

Базовая сигнатура:

```
def __init__(self, *, **kwargs: typing.Any) -> None:
    ...
```

Определение метода `__init__` в классе метрики должно соответствовать следующим правилам:

1. В метод должен быть передан хотя бы один датасет.
2. Каждый параметр должен иметь аннотацию его типа согласно правилам параметризации метрик.
3. Значение параметра по умолчанию не должно нарушать логику валидации указанного типа параметра.
4. Передавать значения параметров в следующие методы нужно через атрибуты экземпляра класса `self`.

>>> **Типы параметров метрик и их использование** <<<

Рекомендуем проводить все проверки введённых значений в рамках данного метода

Пример для метрики "Тест Колмогорова-Смирнова":

```
def __init__(
        self,
        df: pd.DataFrame,
        scale_column: str,
        target_column: str,
        threshold_yellow: float = 10,
        threshold_red: float = 30,
    ):
        self.scale_column = scale_column
        self.target_column = target_column
        self.df = df.astype({self.target_column: "float"})
        self.threshold_yellow = threshold_yellow
        self.threshold_red = threshold_red

        if self.df.empty:
            raise Exception("Dataframe is empty")
        if self.target_column not in self.df:
            raise ValueError(f"Field {self.target_column} does not exist in the dataframe")
        if self.scale_column not in self.df:
            raise ValueError(f"Field {self.scale_column} does not exist in the dataframe")
        if self.df[self.scale_column].nunique() > 100:
            raise Exception("Ошибка: переменная scale не является категориальной")
```

Таким образом, "Тест Колмогорова-Смирнова" принимает на вход:

- `pandas` датафрэйм с данными для расчёта;
- имя колонки со шкалой;
- имя колонки с целевой переменной;
- желтая и красная границы светофоров.

У границ светофора в данном примере есть значение по умолчанию.

В теле метода экземпляру метрики (`self`) присваиваются значения параметров для использования в остальных методах, и проводятся проверки правильности введенных данных.

`__call__`

Метод вызова класса содержит основные расчёты метрики. Обычно в нем расположен блок составления графиков метрики, но ограничений на содержимое нет, метод может содержать любые расчёты

Базовая сигнатура:

```
def __call__(self) -> None:
    ...
```

Метод не имеет параметров, кроме `self` и ничего не возвращает. Все рассчитанные величины должны быть сохранены в атрибуты экземпляра класса `self`.

Пример для метрики "Тест Колмогорова-Смирнова":

```
def __call__(self) -> None:
        dataset = self.df.loc[:, [self.target_column, self.scale_column]].dropna()
```

```
        # номер разряда рейтинговой шкалы (способ получения зависит от формата данных в столбце self.scale)
        dataset["bin_number"] = dataset[self.scale_column].map(
            lambda x: int(x.split("_")[-1])
        )  # dataset[self.scale].astype('category').cat.codes#

        dataset = dataset.sort_values(by=["bin_number"], ascending=False)

        good_cnt = dataset[dataset[self.target_column] == 0].shape[0]
        bad_cnt = dataset[dataset[self.target_column] == 1].shape[0]

        gr_bad = (
            pd.DataFrame(
                dataset.groupby("bin_number", observed=False)[self.target_column].sum()
            ).cumsum()
            / bad_cnt
        )
        dataset["target_inverse"] = np.where(dataset[self.target_column] == 1, 0, 1)
        gr_good = (
            pd.DataFrame(
                dataset.groupby("bin_number", observed=False)["target_inverse"].sum()
            ).cumsum()
            / good_cnt
        )

        ks_calc_temp = pd.merge(gr_good, gr_bad, how="left", left_index=True, right_index=True)
        ks_calc_temp["diff"] = 0
        ks_calc_temp["diff"] = abs(
            ks_calc_temp.iloc[:, 0:1].values - ks_calc_temp.iloc[:, 1:2].values
        )

        self.scalar_value = 100 * ks_calc_temp["diff"].max()
        ks_result = round(self.scalar_value, 2)
        result_idx = ks_calc_temp["diff"].argmax()

        x_value1 = gr_bad.index.values.tolist()
        y_value1 = gr_bad[self.target_column].values.astype("float").tolist()
        x_value2 = gr_good.index.values.tolist()
        y_value2 = gr_good["target_inverse"].values.astype("float").tolist()
        x_value3 = [
            float(gr_bad.index.values[result_idx]),
            float(gr_bad.index.values[result_idx]),
        ]
        y_value3 = [
            float(gr_bad[self.target_column].values[result_idx]),
            float(gr_good["target_inverse"].values[result_idx]),
        ]

        line1 = go.Scatter(
            mode="lines",
            x=x_value1,
            y=y_value1,
            name="bad",
            line={"width": 3},
            marker={"color": "#63666A"},
        )
        line2 = go.Scatter(
            mode="lines",
            x=x_value2,
            y=y_value2,
            name="good",
            line={"width": 3},
            marker={"color": "#3eb489"},
        )
        line3 = go.Scatter(
            mode="lines",
            x=x_value3,
            y=y_value3,
            name=f"KS-statistic = {ks_result.astype('float')}",
            marker={"color": "black"},
        )
        self.fig = go.Figure(data=[line1, line2, line3])
        self.fig.layout = self.custom_layout()
```

Здесь для метрики "Тест Колмогорова-Смирнова" были рассчитаны следующие параметры:

- `self.scalar_value` - скалярное значение для данной метрики;
- `self.fig` - Plotly-график метрики.

Если метрика возвращает исключительно скалярные значения, допустимо пропустить описание данного метода. Так для метрики "Максимальное значение":

```
def __call__(self) -> None:
        pass
```

save

Метод `save` предназначен для сохранения всех графиков в файлы для дальнейшего вывода в отчетах.

Базовая сигнатура:

```
def save(self, output_dir: str) -> dict[str, str] | None:
    ...
```

Метод принимает на вход путь до папки сохранения в строковой переменной `output_dir`. Метод возвращает словарь значений <алиас графика>: <путь до файла>

Predicate умеет обрабатывать данные только следующих форматов:

1. HTML;
2. PNG;
3. JPG;
4. SVG.

Вы **можете** сохранить результаты работы метрики в другом формате, но вам необходимо в этом же методе составить HTML-файл и встроить в него ссылку на ваш файл в неизвестном для Predicate формате. Так система сможет отобразить его в отчете.

Пример метода `save` для метрики "Тест Колмогорова-Смирнова":

```
def save(self, output_dir: str) -> dict[str, str] | None:
        self.fig.write_html(
            f"{output_dir}/data.html",
            config={"displaylogo": False},  # remove the plotly logo
        )
        return {f"scale_{self.scale_column}": f"{output_dir}/data.html"}
```

В данном случае был сохранен один график из атрибута `self.fig`. Но также можно сохранить множество графиков, или сохранять один или множество графиков по условию, как в метрике "Плотность распределения":

```
def save(self, output_dir: str) -> dict[str, str] | None:
        if self.split_charts:
            result = {}
            for column_name, fig in self.figs.items():
                file_path = f"{output_dir}/data_{column_name}.html"
                fig.write_html(
                    file_path,
                    config={"displaylogo": False},  # remove the plotly logo
                )

                result[column_name] = file_path
            return result
        else:
            self.fig.write_html(
                f"{output_dir}/data.html",
                config={"displaylogo": False},  # remove the plotly logo
            )

            return {"fig_name": f"{output_dir}/data.html"}
```

Если метрика не возвращает ни одного графика, допустимо пропустить описание данного метода. Так для метрики "Максимальное значение":

```
def save(self, output_dir: str) -> dict[str, str] | None:
        pass
```

scalar

Данный метод должен появиться в классе метрики в том случае, если флаг `is_scalar = True`.

Метод предназначен для вычисления числового значения метрики.

Базовая сигнатура:

```
def scalar(self) -> int | float:
    ...
```

Метод не принимает на вход никакие параметры. Метод возвращает числовое значение в виде `int` или `float`.

Для нескалярных метрик (флаг `is_scalar = False`) этот метод определять не нужно. Если определить, Predicate воспримет это как ошибку.

Пример метода `scalar` для метрики "Тест Колмогорова-Смирнова":

```
def scalar(self) -> int | float:
        return self.scalar_value
```

В данном случае скалярное значение было рассчитано заранее, и его нужно было только вернуть из метода.

Для метрик без графиков характерно внесение логики расчета числового значения в метод `scalar`. Так для метрики "Максимальное значение":

```
def scalar(self) -> int | float:
        self.scalar_value = float(self.df[self.field_column].min(skipna=True))

        return self.scalar_value
```

`signal`

Данный метод должен появиться в классе метрики в том случае, если флаг `is_signal = True`.

Метод предназначен для вычисления светофора метрики.

Светофор - это индикатор качества метрики. Он выражается в трех цветах:

- Зеленый (`green`) - значения метрики в пределах нормы.
- Желтый (`yellow`) - значения метрики вызывают опасения.
- Красный (`red`) - значения метрики далеко за пределами нормы.

Базовая сигнатура:

```
def signal(self) -> Literal["red", "yellow", "green"]:
    ...
```

Метод не принимает на вход никакие параметры. Метод возвращает строковое значение светофора. Значения светофора должны быть строго из списка `["red", "yellow", "green"]`.

Пример метода `signal` для метрики "Тест Колмогорова-Смирнова":

```
def signal(self) -> Literal["red", "yellow", "green"]:
        signal_light = "green"

        if self.scalar_value > self.threshold_red:
            signal_light = "red"
        elif self.scalar_value > self.threshold_yellow:
            signal_light = "yellow"

        return signal_light
```

ПОРЯДОК ИСПОЛНЕНИЯ МЕТОДОВ

Методы и в продукте Predicate, и в библиотеке Predicate исполняются в следующем порядке:

1. `__init__`;
2. `__call__`;
3. `save`;
4. `scalar` (если флаг `is_scalar = True`);
5. `signal` (если флаг `is_signal = True`).

Учитывайте порядок исполнения методов при разработке метрики. От него зависит, как можно передавать значения из метода в метод.

Рекомендации по написанию метрик

Кроме явных требований к структуре класса метрики, есть ряд необязательных рекомендаций к их написанию. Эти рекомендации упрощают работу с метрикой: ее поддержку разными членами команды; создание множества метрик по аналогичному шаблону; проверку данных, введенных в метрику.

Список рекомендаций не конечный. Мы будем увеличивать его при появлении новых идей.

МЕТОД `custom_layout`

Если ваша метрика возвращает график в виде объекта **Plotly**, то мы рекомендуем создавать метод `custom_layout`.

Layout - это макета графика. В layout описываются параметры представления графика: заголовок; названия осей; границы отображаемых осей; и можество других параметров. Подробнее об этом - в документации Plotly.

Layout обычно занимает от десяти строчек кода. Метод `custom_layout` позволяет вам не загромождать метод `__call__` лишними описаниями, а вынести их в отдельный метод.

Метод `custom_layout` упрощает копирование кода из метрики в метрику. В вашем описании графика Plotly будет фигурировать только обращение к методу `custom_layout`, поэтому вы сможете перенести этот график в другую метрику с другим layout-ом без дополнительного переписывания кода.

Пример метода `custom_layout` для метрики "Тест Колмогорова-Смирнова":

```
def custom_layout(self) -> Optional[Dict[str, Any]]:
        return {
            "title": {"text": "<b>Тест Колмогорова-Смирнова</b>", "x": 0.1, "y": 0.97},
            "legend": {"yanchor": "bottom", "y": 0.05, "xanchor": "right", "x": 1},
            "yaxis": {"title": "Кумулятивная доля", "side": "left"},
            "xaxis": {
                "title": "Разряд рейтинговой шкалы",
                "side": "left",
                "type": "category",
                "domain": [0, 0.8],
            },
            "margin": {"t": 35, "b": 5, "l": 5, "r": 5},
        }
```

ПОЛЕ `scalar_value`

Мы рекомендуем присваивать числовое значение метрики в атрибут `scalar_value`. Это поле упрощает жизнь: становится проще читать код; всегда понятно, что возвращать в методе `scalar`; легко обратиться к числовому значению в других методах.

2.3.4 Параметры метрик Predicate

№	Название типа	Параметризация типа
1	Датасет	`pandas.DataFrame`
2	Строка	1. Тип: `str` 2. В имени параметра не содержится суффикс `_column`
3	Целое число	`int`
4	Вещественное число	`float`
5	Диапазон из двух вещественных чисел	`Tuple[float, float]`
6	Логическая переменная	`bool`
7	Дропдаун из заданных пользователем значений	`Literal["option1", "option2", "option3"]`
8	Дропдаун с мультивыбором из заданных пользователем значений	`List[Literal["option1", "option2", "option3"]`
9	Дропдаун из столбцов датасета	1. Тип: `str` 2. В имени параметра содержится суффикс `_column`
10	Дропдаун с мультивыбором из столбцов датасета	1. Тип: `List[str]` 2. В имени параметра содержится суффикс `_columns`
11	Дата	`datetime.date`
12	Время	`datetime.time`

2.3.5 Примеры кода метрик

Просмотр python-кода имеющихся в системе метрик доступен из интерфейса приложения:

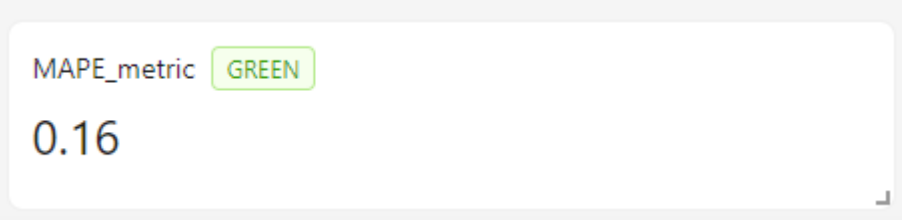
1. На странице каталога метрик (*Панель управления > Метрики*) выберите нужную метрику, наведите курсор на символ меню (три точки) в правой части соответствующей строки и нажмите “Просмотр”.
2. Откроется экранная форма с информацией о метрике. Прокрутите страницу вниз и нажмите кнопку "Открыть код" в правом нижнем углу экранной формы.

ПРИМЕР СКАЛЯРНОЙ МЕТРИКИ СО СВЕТОФОРОМ БЕЗ ГРАФИКА

Указаны флаги `is_scalar = True` и `is_signal = True`. Методы `scalar` и `signal` объявлены и имплементированы. Методы `__call__` и `save` объявлены, но имплементация пропущена.

```python
from typing import Literal

import pandas as pd
from sklearn.metrics import mean_absolute_percentage_error


class rvc_3_MAPE:
    """
    Средняя абсолютная ошибка в процентах

    Attributes:
        __desc__ (str): Description of the class.
        __tags__ (list[str]): List of tags associated with the class.
        is_scalar (bool): Whether the metric is scalar or not.
        is_signal (bool): Whether the metric has signal or not.

    """

    __desc__ = "Mean Absolute Percentage Error (MAPE). Средняя абсолютная ошибка в процентах"
    __tags__ = ["core", "regression", "scalar"]
    is_scalar = True
    is_signal = True

    def __init__(
        self,
        df: pd.DataFrame,
        predict_column: str,
        target_column: str,
        threshold_yellow: float = 0.3,
        threshold_red: float = 0.4,
    ):
        if df.empty:
            raise Exception("Dataframe is empty")
        if target_column not in df:
            raise ValueError(f"Field {target_column} does not exist in the dataframe")
        if predict_column not in df:
            raise ValueError(f"Field {predict_column} does not exist in the dataframe")

        self.predict_column = predict_column
        self.target_column = target_column
        self.df = df.astype({self.predict_column: "float", self.target_column: "float"})
        self.threshold_yellow = threshold_yellow
        self.threshold_red = threshold_red

    def __call__(self) -> None:
        pass

    def scalar(self) -> int | float:
        df = self.df.loc[:, [self.target_column, self.predict_column]].dropna()[
            abs(self.df[self.target_column]) > 0
        ]

        self.scalar_value = mean_absolute_percentage_error(
            y_pred=df[self.predict_column],
            y_true=df[self.target_column],
        )

        return self.scalar_value

    def signal(self) -> Literal["red", "yellow", "green"]:
        signal_light = "green"

        if self.scalar_value > self.threshold_red:
            signal_light = "red"
        elif self.scalar_value > self.threshold_yellow:
            signal_light = "yellow"

        return signal_light

    def save(self, output_dir: str) -> dict[str, str] | None:
        pass
```

Результат:



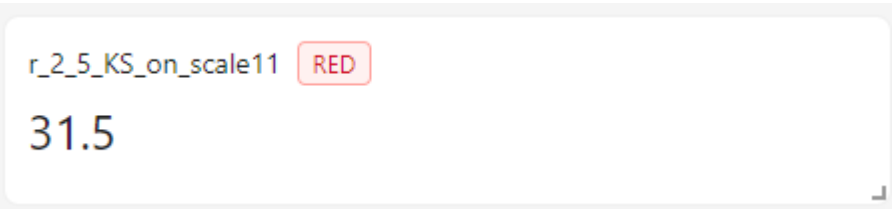
ПРИМЕР СКАЛЯРНОЙ МЕТРИКИ СО СВЕТОФОРОМ И С ГРАФИКОМ

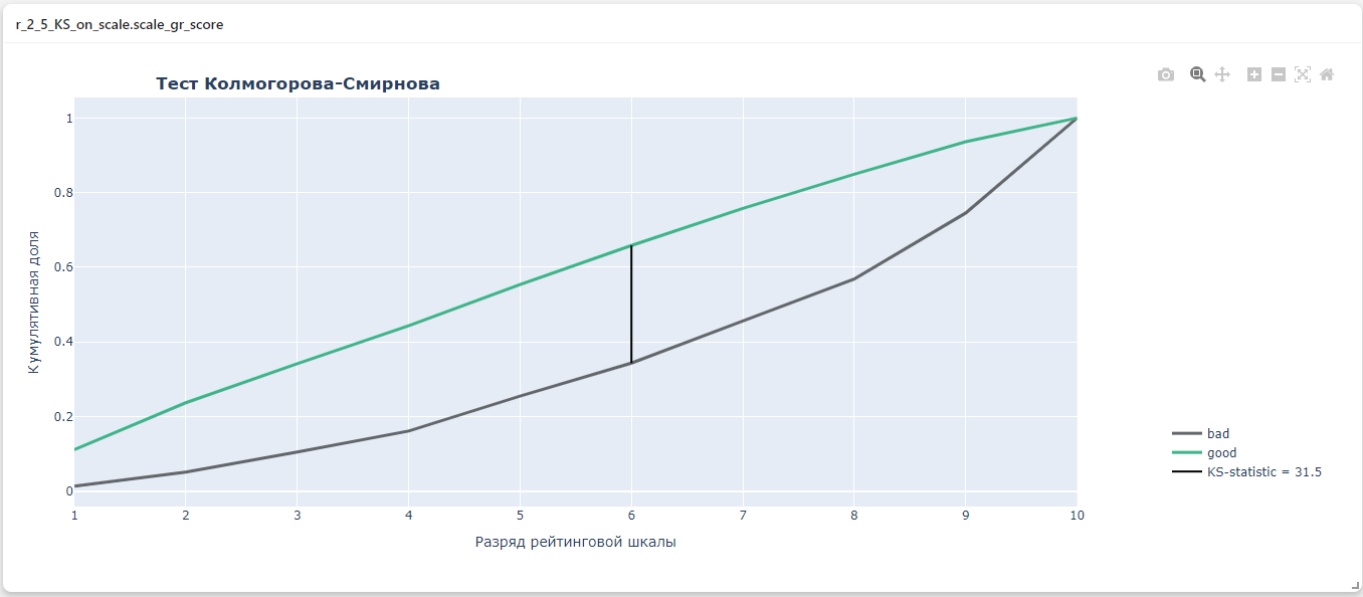
Указаны флаги `is_scalar = True` и `is_signal = True`. Методы `scalar` и `signal` объявлены и имплементированы. Методы `__call__` и `save` объявлены и имплементированы.

```
from typing import Any, Dict, Literal, Optional

import numpy as np
import pandas as pd
import plotly.graph_objects as go


class r_2_5_KS_on_scale:
    """
    Тест Колмогорова-Смирнова

    Показывает насколько хорошо score модели
    отделяет "хороших" клиентов от "плохих" в разрезе рейтинговой шкалы.

    Attributes:
        __desc__ (str): Description of the class.
        __tags__ (list[str]): List of tags associated with the class.
        is_scalar (bool): Whether the metric is scalar or not.
        is_signal (bool): Whether the metric has signal or not.

    """

    __desc__ = "KS-test on scale. Тест Колмогорова-Смирнова"
    __tags__ = ["risk", "scalar"]

    is_scalar = True
    is_signal = True

    def __init__(
        self,
        df: pd.DataFrame,
        scale_column: str,
        target_column: str,
        threshold_yellow: float = 10,
        threshold_red: float = 30,
    ):
        self.scale_column = scale_column
        self.target_column = target_column
        self.df = df.astype({self.target_column: "float"})
        self.threshold_yellow = threshold_yellow
        self.threshold_red = threshold_red

        if self.df.empty:
            raise Exception("Dataframe is empty")
        if self.target_column not in self.df:
            raise ValueError(f"Field {self.target_column} does not exist in the dataframe")
        if self.scale_column not in self.df:
            raise ValueError(f"Field {self.scale_column} does not exist in the dataframe")
        if self.df[self.scale_column].nunique() > 100:
            raise Exception("Ошибка: переменная scale не является категориальной")

    def __call__(self) -> None:
        dataset = self.df.loc[:, [self.target_column, self.scale_column]].dropna()

        # номер разряда рейтинговой шкалы
        # (способ получения зависит от формата данных в столбце self.scale)
        dataset["bin_number"] = dataset[self.scale_column].map(
            lambda x: int(x.split("_")[-1])
        )  # dataset[self.scale].astype('category').cat.codes#

        dataset = dataset.sort_values(by=["bin_number"], ascending=False)

        good_cnt = dataset[dataset[self.target_column] == 0].shape[0]
        bad_cnt = dataset[dataset[self.target_column] == 1].shape[0]

        gr_bad = (
            pd.DataFrame(
                dataset.groupby("bin_number", observed=False)[self.target_column].sum()
            ).cumsum()
            / bad_cnt
        )
        dataset["target_inverse"] = np.where(dataset[self.target_column] == 1, 0, 1)
        gr_good = (
            pd.DataFrame(
                dataset.groupby("bin_number", observed=False)["target_inverse"].sum()
            ).cumsum()
            / good_cnt
        )

        ks_calc_temp = pd.merge(gr_good, gr_bad, how="left", left_index=True, right_index=True)
        ks_calc_temp["diff"] = 0
        ks_calc_temp["diff"] = abs(
            ks_calc_temp.iloc[:, 0:1].values - ks_calc_temp.iloc[:, 1:2].values
        )

        self.scalar_value = 100 * ks_calc_temp["diff"].max()
        ks_result = round(self.scalar_value, 2)
```

```
        result_idx = ks_calc_temp["diff"].argmax()

        x_value1 = gr_bad.index.values.tolist()
        y_value1 = gr_bad[self.target_column].values.astype("float").tolist()
        x_value2 = gr_good.index.values.tolist()
        y_value2 = gr_good["target_inverse"].values.astype("float").tolist()
        x_value3 = [
            float(gr_bad.index.values[result_idx]),
            float(gr_bad.index.values[result_idx]),
        ]
        y_value3 = [
            float(gr_bad[self.target_column].values[result_idx]),
            float(gr_good["target_inverse"].values[result_idx]),
        ]

        line1 = go.Scatter(
            mode="lines",
            x=x_value1,
            y=y_value1,
            name="bad",
            line={"width": 3},
            marker={"color": "#63666A"},
        )
        line2 = go.Scatter(
            mode="lines",
            x=x_value2,
            y=y_value2,
            name="good",
            line={"width": 3},
            marker={"color": "#3eb489"},
        )
        line3 = go.Scatter(
            mode="lines",
            x=x_value3,
            y=y_value3,
            name=f"KS-statistic = {ks_result.astype('float')}",
            marker={"color": "black"},
        )
        self.fig = go.Figure(data=[line1, line2, line3])
        self.fig.layout = self.custom_layout()

    def scalar(self) -> int | float:
        return self.scalar_value

    def signal(self) -> Literal["red", "yellow", "green"]:
        signal_light = "green"

        if self.scalar_value > self.threshold_red:
            signal_light = "red"
        elif self.scalar_value > self.threshold_yellow:
            signal_light = "yellow"

        return signal_light

    def custom_layout(self) -> Optional[Dict[str, Any]]:
        return {
            "title": {"text": "<b>Тест Колмогорова-Смирнова</b>", "x": 0.1, "y": 0.97},
            "legend": {"yanchor": "bottom", "y": 0.05, "xanchor": "right", "x": 1},
            "yaxis": {"title": "Кумулятивная доля", "side": "left"},
            "xaxis": {
                "title": "Разряд рейтинговой шкалы",
                "side": "left",
                "type": "category",
                "domain": [0, 0.8],
            },
            "margin": {"t": 35, "b": 5, "l": 5, "r": 5},
        }

    def save(self, output_dir: str) -> dict[str, str] | None:
        self.fig.write_html(
            f"{output_dir}/data.html",
            config={"displaylogo": False},  # remove the plotly logo
        )
        return {f"scale_{self.scale_column}": f"{output_dir}/data.html"}
```

Результат:





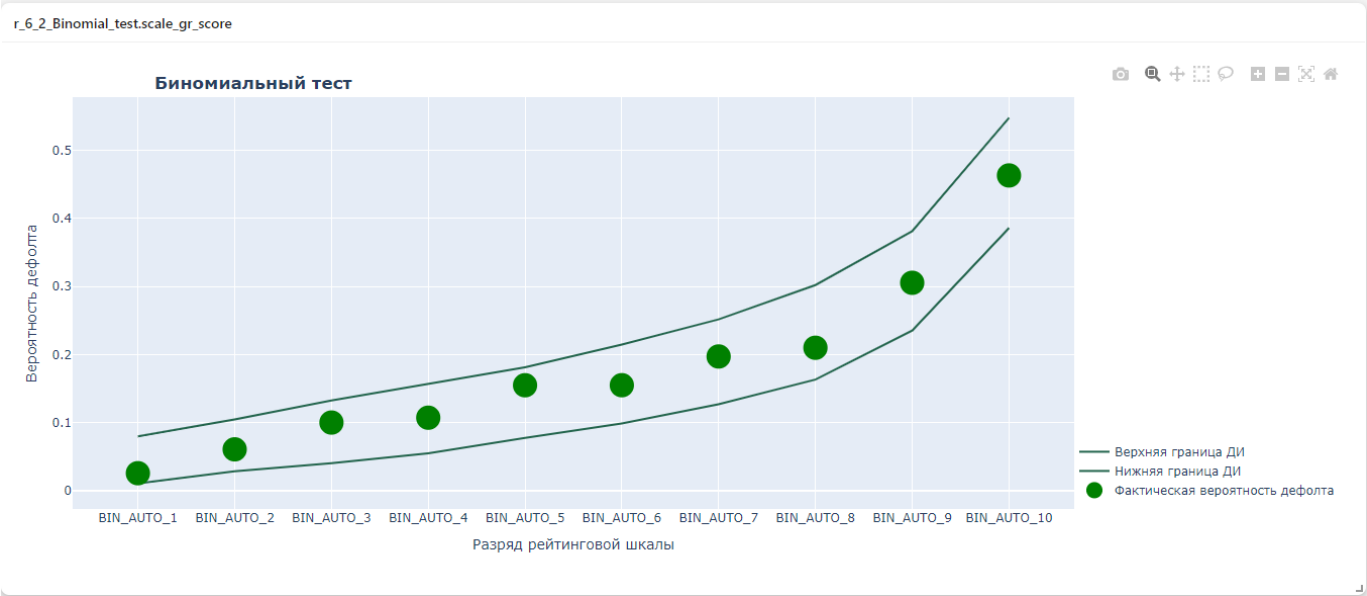
ПРИМЕР КОДА МЕТРИКИ С ГРАФИКОМ, НО БЕЗ СКАЛЯРА И СВЕТОФОРА

Указаны флаги `is_scalar = False` и `is_signal = False`. Методы `scalar` и `signal` не объявлены. Методы `__call__` и `save` объявлены и имплементированы.

```
from typing import Any, Dict, Optional

import numpy as np
import pandas as pd
import plotly.graph_objects as go
from scipy.stats import norm


class r_6_2_Binomial_test:
    """
    Проверяет попадение среднего уровня дефолта по бакету рейтинговой шкалы в
    доверительный интервал, построенный по скорам модели

    Attributes:
        __desc__ (str): Description of the class.
        __tags__ (list[str]): List of tags associated with the class.
        is_scalar (bool): Whether the metric is scalar or not.
        is_signal (bool): Whether the metric has signal or not.

    """

    __desc__ = "Binomial Test. Биномиальный тест"
    __tags__ = ["risk"]

    is_scalar = False
    is_signal = False

    def __init__(
        self,
        df: pd.DataFrame,
        predict_column: str,
        target_column: str,
        scale_column: str,
        confidence_level: float = 0.99,
    ):
        if predict_column not in df.columns:
            raise ValueError(
                f"Invalid column name for 'predict_column'. "
                f"There is not colomn '{predict_column}' in the dataframe"
            )
        if target_column not in df.columns:
            raise ValueError(
                f"Invalid column name for 'target_column'. "
                f"There is not colomn '{target_column}' in the dataframe"
            )
        if scale_column not in df.columns:
            raise ValueError(
                f"Invalid column name for 'scale_column'. "
                f"There is not colomn '{scale_column}' in the dataframe"
            )

        self.predict_column = predict_column
        self.target_column = target_column
        self.scale_column = scale_column
        self.df = df.astype({self.predict_column: "float", self.target_column: "float"})
```

```
        self.confidence_level = confidence_level

        if self.df.empty:
            raise ValueError("Dataframe is empty")
        if self.df[self.scale_column].nunique() > 100:
            raise Exception("Ошибка: переменная scale не является категориальной")

    def __call__(self) -> None:
        data_gr = (
            self.df[[self.scale_column, self.target_column, self.predict_column]]
            .groupby([self.scale_column], observed=False)
            .agg({self.target_column: ["sum", "count"], self.predict_column: ["mean"]})
            .reset_index()
        )

        data_gr.columns = [self.scale_column, self.target_column, "cnt_all", self.predict_column]
        data_gr["target_prc"] = data_gr[self.target_column] / data_gr["cnt_all"]

        data_gr["CI_LEFT"] = data_gr[self.predict_column] - norm.ppf(
            self.confidence_level
        ) * np.sqrt(
            (data_gr[self.predict_column] * (1 - data_gr[self.predict_column])) / data_gr["cnt_all"]
        )
        data_gr["CI_RIGHT"] = data_gr[self.predict_column] + norm.ppf(
            self.confidence_level
        ) * np.sqrt(
            (data_gr[self.predict_column] * (1 - data_gr[self.predict_column])) / data_gr["cnt_all"]
        )

        data_gr["color"] = data_gr.apply(
            lambda x: "green"
            if (x["target_prc"] >= x["CI_LEFT"]) & (x["target_prc"] <= x["CI_RIGHT"])
            else "red",
            axis=1,
        )

        data_gr = data_gr.sort_values(self.scale_column, key=lambda x: x.str[-3:])
        # упорядочивание выше - под конкретный df,
        # обращать внимание на формат записей в столбце scale
        # при запуске на новых данных

        line1 = go.Scatter(
            mode="lines",
            x=data_gr[self.scale_column].tolist(),
            y=data_gr["CI_RIGHT"].tolist(),
            name="Верхняя граница ДИ",
            marker={"color": "#23654D"},
            xaxis="x1",
            yaxis="y1",
        )
        line2 = go.Scatter(
            mode="lines",
            x=data_gr[self.scale_column].tolist(),
            y=data_gr["CI_LEFT"].tolist(),
            name="Нижняя граница ДИ",
            marker={"color": "#23654D"},
            xaxis="x1",
            yaxis="y1",
        )
        line3 = go.Scatter(
            mode="markers",
            x=data_gr[self.scale_column].tolist(),
            y=data_gr["target_prc"].tolist(),
            name="Фактическая вероятность дефолта",
            marker={"color": data_gr["color"].tolist(), "size": 24},
            xaxis="x1",
            yaxis="y1",
        )

        self.fig = go.Figure(data=[line1, line2, line3])
        self.fig.layout = self.custom_layout()

    def custom_layout(self) -> Optional[Dict[str, Any]]:
        return {
            "title": {"text": "<b>Биномиальный тест</b>", "x": 0.1, "y": 0.97},
            "legend": {"yanchor": "bottom", "y": 0.01, "xanchor": "left", "x": 1},
            "yaxis": {"title": "Вероятность дефолта", "side": "left"},
            "xaxis": {
                "title": "Разряд рейтинговой шкалы",
                "side": "right",
                "type": "category",
                "domain": [0, 1],
            },
            "margin": {"t": 35, "b": 5, "l": 5, "r": 5},
        }

    def save(self, output_dir: str) -> dict[str, str] | None:
        self.fig.write_html(
            f"{output_dir}/data.html",
            config={"displaylogo": False},  # remove the plotly logo
        )
        return {f"scale_{self.scale_column}": f"{output_dir}/data.html"}
```

Результат:



ПРИМЕР КОДА МЕТРИКИ С МНОЖЕСТВОМ ГРАФИКОВ

Методы `__call__` и `save` объявлены и имплементированы.

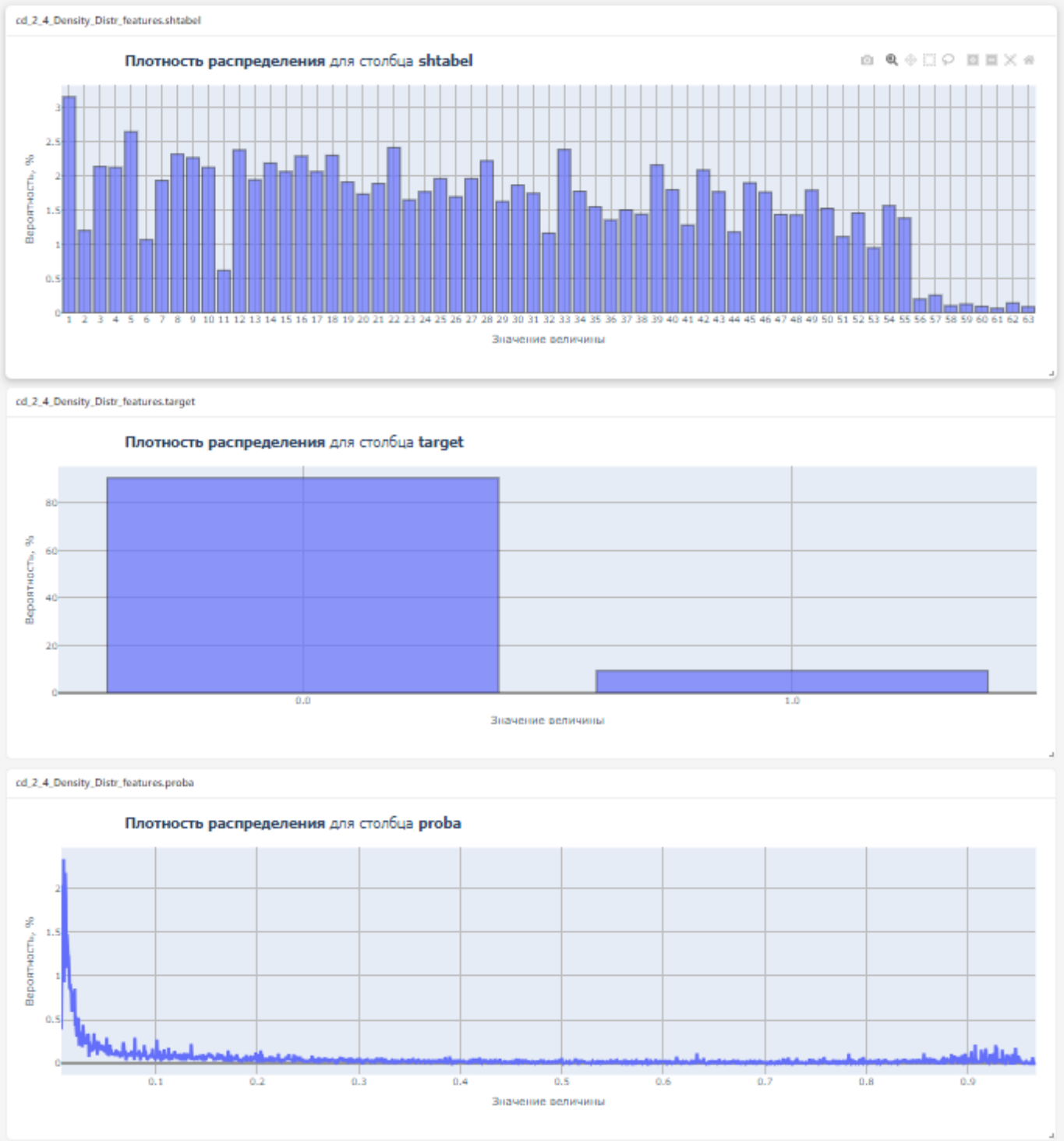
В методе `save` графики создаются в цикле

```
from typing import Any, Dict, Optional

import pandas as pd
import plotly.graph_objects as go


class cd_2_4_Density_Distr_features:
    """
    Плотность распределения для выбранных полей
    Attributes:
        __desc__ (str): Description of the class.
        __tags__ (list[str]): List of tags associated with the class.
        is_scalar (bool): Whether the metric is scalar or not.
        is_signal (bool): Whether the metric has signal or not.

    """

    __desc__ = (
        "Density Distribution for Selected Columns. Плотность распределения для выбранных полей"
    )
    __tags__ = ["core", "data"]

    is_scalar = False
    is_signal = False

    def __init__(
        self,
        df: pd.DataFrame,
        field_columns: str,
        categorial_threshold: int = 10,
        split_charts: bool = False,
    ):
        self.df = df
        self.categorial_threshold = categorial_threshold
        self.split_charts = split_charts
        self.field_columns = [x.strip() for x in field_columns.split(",")]
        if self.df.empty:
            raise Exception("Dataframe is empty")
        for field in self.field_columns:
            if field not in self.df:
                raise ValueError(f"Field {field} does not exist in the dataframe")

    def __call__(self) -> None:
        self.df = self.df[self.field_columns]
        charts_dict = self.create_fields_charts()

        if self.split_charts:
            self.figs = {
                column_name: go.Figure(
                    data=[chart], layout=self.custom_layout(column_name=column_name)
                )
```

```
                for column_name, chart in charts_dict.items()
            }
        else:
            self.fig = go.Figure(data=list(charts_dict.values()), layout=self.custom_layout())

    def create_fields_charts(self):
        signal = {}
        self.min_x = 0
        self.max_x = 0
        counted_labels = []

        # цикл по всем столбцам df
        for columnName, columnData in self.df.items():
            # если данные в столбце не числовые, пропускаем его
            if not pd.api.types.is_numeric_dtype(columnData):
                print(f'Column "{columnName}" type is not numeric')
                continue

            counted_labels.append(columnName)
            visible_mode = (
                "legendonly" if columnName != counted_labels[0] and not self.split_charts else True
            )

            # если данные в столбце категориальные, строим гистограмму
            if columnData.nunique() <= self.categorial_threshold:
                freq_df = (
                    columnData.value_counts(normalize=True, sort=False, dropna=True)
                    .reset_index()
                    .sort_values(columnName)
                )

                freq_df["percent"] = freq_df["proportion"] * 100
                if self.split_charts:
                    freq_df[columnName] = freq_df[columnName].astype("string")

                elem = go.Bar(
                    x=freq_df[columnName].tolist(),
                    y=freq_df["percent"].tolist(),
                    name=columnName,
                    opacity=0.7,
                    marker=dict(line=dict(color="black", width=1.0)),
                    visible=visible_mode,
                )
                signal[columnName] = elem
                continue

            # иначе - линейный график плотности распределения
            vals = columnData.dropna().values
            nbucket = int(len(vals) / 10) + 1
            den_x = []
            den_y = []
            wgth = (max(vals) - min(vals)) / nbucket  # ширина одного интервала
            minval = min(vals)
            self.max_x = max(max(vals), self.max_x)
            self.min_x = min(min(vals), self.min_x)
            self.max_pos = 0

            for i in range(0, nbucket):
                count = 0
                for j in vals:
                    if (minval + i * wgth) <= j < (minval + (i * wgth) + wgth):
                        count = count + 1
                den_x.append(round((minval + i * wgth + wgth / 2), 6))
                den_y.append(round(count * 100 / (len(vals)), 6))

            elem = go.Scatter(
                x=den_x,
                y=den_y,
                name=columnName,
                mode="lines",
                line_width=4,
                line_dash="solid",
                visible=visible_mode,
            )

            self.max_pos = max(max(den_y), self.max_pos)

            signal[columnName] = elem

        return signal

    def custom_layout(self, column_name: str | None = None) -> Optional[Dict[str, Any]]:
        column_info = (
            " избранных столбцов" if column_name is None else f" для столбца <b>{column_name}</b>"
        )
        return {
            "title": {"text": f"<b>Плотность распределения</b>{column_info}", "x": 0.1, "y": 0.98},
            "legend": {"yanchor": "bottom", "y": 0.05, "xanchor": "right", "x": 1},
            "xaxis": {
                "title": "Значение величины",
                "side": "left",
                "showgrid": True,
                "zeroline": True,
                "gridcolor": "#bdbdbd",
```

```
                "gridwidth": 1.5,
                "zerolinecolor": "#969696",
                "zerolinewidth": 3,
            },
            "yaxis": {
                "title": "Вероятность, %",
                "side": "left",
                "showgrid": True,
                "zeroline": True,
                "gridcolor": "#bdbdbd",
                "gridwidth": 1.5,
                "zerolinecolor": "#969696",
                "zerolinewidth": 3,
            },
            "margin": {"t": 45, "b": 5, "l": 5, "r": 5},
        }

    def save(self, output_dir: str) -> dict[str, str] | None:
        if self.split_charts:
            result = {}
            for column_name, fig in self.figs.items():
                file_path = f"{output_dir}/data_{column_name}.html"
                fig.write_html(
                    file_path,
                    config={"displaylogo": False},  # remove the plotly logo
                )

                result[column_name] = file_path
            return result
        else:
            self.fig.write_html(
                f"{output_dir}/data.html",
                config={"displaylogo": False},  # remove the plotly logo
            )

            return {"fig_name": f"{output_dir}/data.html"}
```

Результат:



ПРИМЕР КОДА МЕТРИКИ, СОХРАНЯЮЩЕЙ РЕЗУЛЬТАТ В ВИДЕ КАРТИНКИ

Методы `__call__` и `save` объявлены и имплементированы.

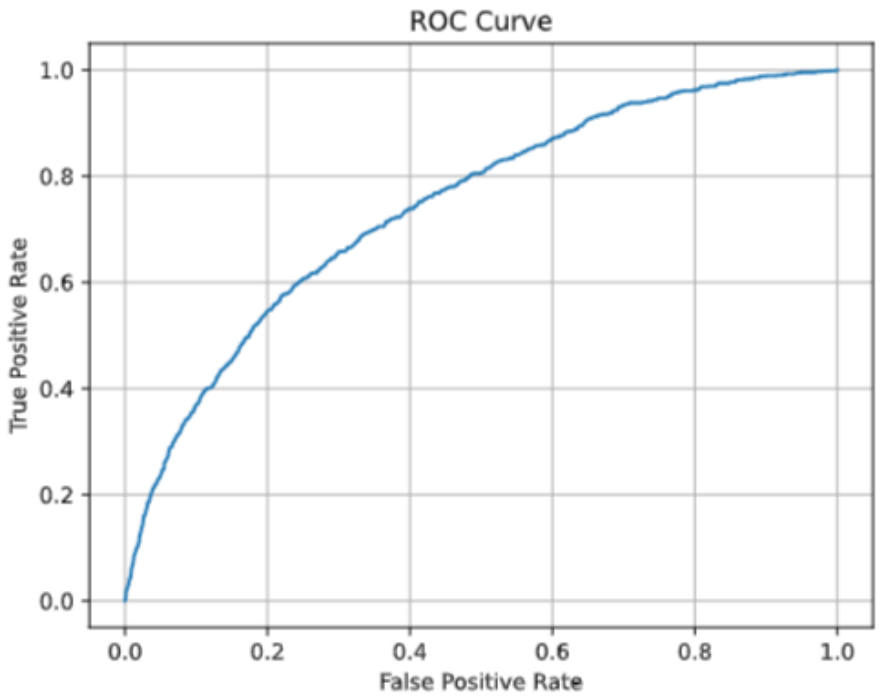
В методе `save` графики сохраняются как картинки, а не HTML-файлы

```
from typing import Any, Dict, Literal, Optional

import numpy as np
import pandas as pd
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt


class ROC_AUC_img:
    """
```

```
    Значение ROC-AUC и График ROC Curve

    Attributes:
        __desc__ (str): Description of the class.
        __tags__ (list[str]): List of tags associated with the class.
        is_scalar (bool): Whether the metric is scalar or not.
        is_signal (bool): Whether the metric has signal or not.

    """

    __desc__ = "График ROC Curve, значение ROC-AUC"
    __tags__ = ["core", "classification", "scalar"]

    is_scalar = True
    is_signal = True

    def __init__(
        self,
        df: pd.DataFrame,
        predict_column: str,
        target_column: str,
        threshold_yellow: float = 0.75,
        threshold_red: float = 0.65,
    ):
        self.predict_column = predict_column
        self.target_column = target_column
        self.df = df.astype({self.predict_column: "float", self.target_column: "float"})
        self.threshold_yellow = threshold_yellow
        self.threshold_red = threshold_red

        if self.df.empty:
            raise Exception("Dataframe is empty")
        if self.target_column not in self.df:
            raise ValueError(f"Field {self.target_column} does not exist in the dataframe")
        if self.predict_column not in self.df:
            raise ValueError(f"Field {self.predict_column} does not exist in the dataframe")
        if self.predict_column == self.target_column:
            raise Exception("Ошибка. Проверьте выбор столбцов для расчета")

    def __call__(self) -> None:
        temp = self.df.loc[:, [self.target_column, self.predict_column]].dropna()
        preds = temp[self.predict_column]
        y_test = temp[self.target_column]

        fpr, tpr, threshold = roc_curve(y_test, preds)
        fpr = np.around(fpr, decimals=4).tolist()
        tpr = np.around(tpr, decimals=4).tolist()
        base_roc = np.around(np.linspace(0, 1, 10), decimals=2).tolist()

        self.scalar_value = float(
            roc_auc_score(temp[self.target_column], temp[self.predict_column])
        )
        self.fig, ax = plt.subplots()
        ax.plot(fpr, tpr)
        ax.set(xlabel='False Positive Rate', ylabel='True Positive Rate', title='ROC Curve')
        ax.grid()

    def scalar(self) -> int | float:
        return self.scalar_value

    def signal(self) -> Literal["red", "yellow", "green"]:
        signal_light = "green"

        if self.scalar_value < self.threshold_red:
            signal_light = "red"
        elif self.scalar_value < self.threshold_yellow:
            signal_light = "yellow"

        return signal_light

    def save(self, output_dir: str) -> dict[str, str] | None:
        self.fig.savefig(f"{output_dir}/data.svg")
        return {"svg": f"{output_dir}/data.svg"}
```

Результат:

ROC_AUC_img.svg



2.4 Преобразования

2.4.1 Регистрация нового преобразования

Predicate поддерживает работу с преобразованиями, которые написаны на языке Python.

Загрузка преобразования в систему осуществляется из файла с расширением `.py`.

В базовой версии приложения используются следующие версии языка и пакетов:

Развернуть перечень пакетов

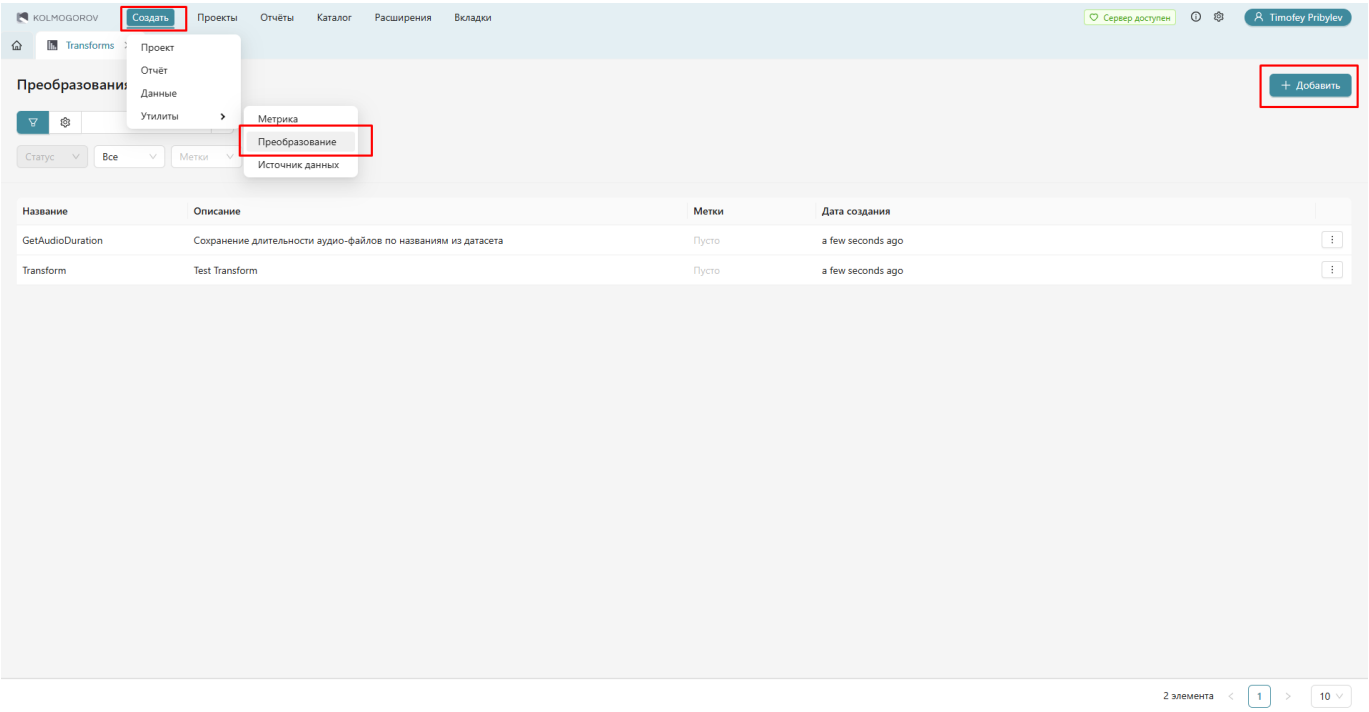
Python версии 3.11
pandas версии 2.2.0
nympy версии 1.26.4
plotly версии 5.20.0
matplotlib версии не ниже 3.9.2
scikit-learn версии не ниже 1.2.0
statsmodels версии не ниже 0.13.5
river версии не ниже 0.16.0

Использование других версий языка и пакетов и добавление новых пакетов возможно, но требует дополнительной настройки приложения. Свяжитесь с администраторами системы для проведения нужных работ.

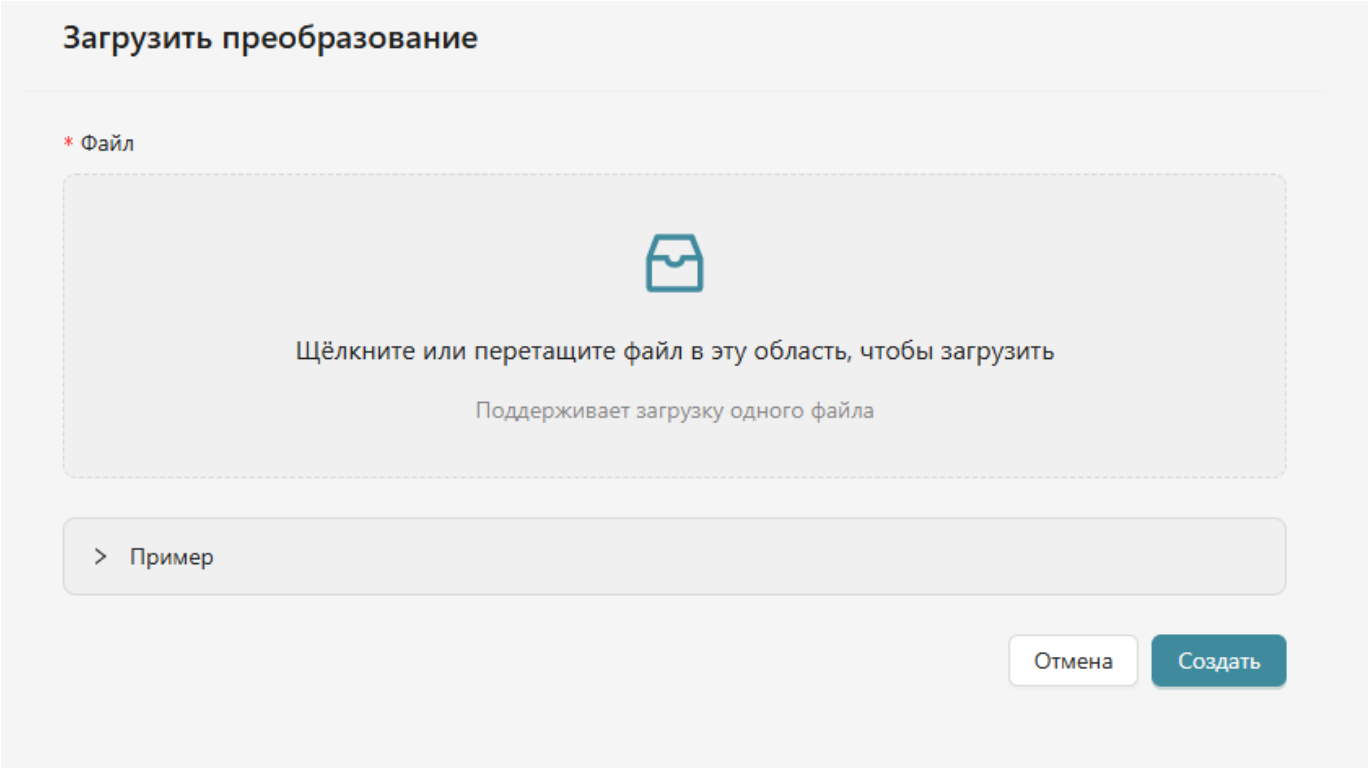
Подробнее о составлении кода преобразований см. на странице "Правила написания преобразований".

Регистрация преобразования

Для загрузки новой метрики перейдите по пути *Панель управления* > *Каталог* > *Преобразования* в основном меню приложения и на открывшейся странице каталога преобразований нажмите кнопку "Добавить". Также добавление преобразования доступно по пути *Панель управления* > *Создать* > *Утилиты* > *Преобразование*.



Откроется форма загрузки нового преобразования:



В открывшейся форме необходимо загрузить python-файл с кодом преобразования и нажать кнопку "Создать".

После того как преобразование создано, соответствующая ему запись появится в списке преобразований, зарегистрированных в системе (*Каталог > Преобразования*), и преобразование станет доступным для использования в проектах мониторинга.

2.4.2 Правила написания преобразований

Преобразование Predicate - это класс Python с определенной структурой. Данный класс не содержит специфичных зависимостей, кроме `pandas`.

На данной странице описана структура класса с преобразованием. В дальнейшем возможно дополнение страницы рекомендациями по написанию преобразований.

Структура класса с преобразованием

У класса преобразования есть набор обязательных полей и методов. При этом нет ограничений на использование дополнительных полей или методов. Мы даже посоветуем использовать дополнительные методы и поля в разделе "Рекомендации по написанию преобразований".

Обязательные поля:

1. `__desc__`;
2. `__tags__`;
3. `__add_column__`;
4. `__del_column__`.

Обязательные методы:

1. `__init__`;
2. `__call__`.

В разделах ниже поля и методы описаны подробнее.

ПОЛЯ

Название класса становится названием преобразования в системе

`__desc__` - понятное человеку описание преобразования.

`__tags__` - список тегов, присвоенных преобразованию в системе.

`__add_column__` - список названий столбцов, которые преобразование добавляет к исходному датасету.

`__del_column__` - список названий столбцов, которые преобразование удаляет из исходного датасета.

Таким образом, поля для преобразования, которое добавляет в датасет информацию о длительности аудиофайлов и ничего не удаляет из датасета, выглядят следующим образом:

```
class GetAudioDuration:
    __desc__ = "Сохранение длительности аудио-файлов по названиям из датасета"
    __tags__ = ["audio"]

    __add_column__ = ["duration_seconds"]
    __del_column__ = []
```

МЕТОДЫ

`__init__`

Метод инициализации класса предназначен для ввода параметров преобразования. Именно в этот метод будут передаваться все параметры преобразования при вызове преобразования в проекте Predicate или в рамках библиотеки.

Базовая сигнатура:

```
def __init__(self, *, **kwargs: typing.Any) -> None:
    ...
```

Определение метода `__init__` в классе преобразования должно соответствовать следующим правилам:

1. В метод должен быть передан хотя бы один датасет.
2. Каждый параметр должен иметь аннотацию его типа согласно правилам параметризации, аналогичным правилам параметризации метрик.
3. Значение параметра по умолчанию не должно нарушать логику валидации указанного типа параметра.
4. Передавать значения параметров в следующие методы нужно через атрибуты экземпляра класса `self`.

Рекомендуем проводить все проверки введённых значений в рамках данного метода

Пример для аудио-преобразования:

```
def __init__(self, df: pd.DataFrame, path_colomn:str = 'path'):
        self.df = df
        self.path_colomn = path_colomn

        if self.df.empty:
            raise Exception("Dataframe is empty")
        if self.path_colomn not in self.df:
            raise ValueError(f"Field {self.path_colomn} does not exist in the dataframe")
```

Таким образом, преобразование принимает на вход:

- `pandas` датафрэйм с данными для расчёта;
- имя колонки с информацией о пути до аудиофайла.

В теле метода экземпляру преобразования (`self`) присваиваются значения параметров для использования в остальных методах, и проводятся проверки правильности введенных данных.

`__call__`

Метод вызова класса содержит основные расчёты преобразования. Ограничений на содержимое нет, метод может содержать любые расчёты.

Базовая сигнатура:

```
def __call__(self) -> pd.DataFrame:
    ...
```

Метод не имеет параметров, кроме `self`, и должен возвращать `pandas.DataFrame`.

Пример для аудио-преобразования:

```
def __call__(self) -> pd.DataFrame:
        durations = [self.get_audio_duration(f'/tmp/data/audio/{path}') for path in self.df[self.path_colomn]]
        self.df['duration_seconds'] = durations

        return self.df
```

Здесь для преобразования был обновлен исходный датафрейм, в который добавили новую колонку `duration_seconds`.

ПОРЯДОК ИСПОЛНЕНИЯ МЕТОДОВ

Методы и в продукте Predicate, и в библиотеке Predicate исполняются в следующем порядке:

1. `__init__`;
2. `__call__`.

Учитывайте порядок исполнения методов при разработке преобразования. От него зависит, как можно передавать значения из метода в метод.

2.5 Проекты

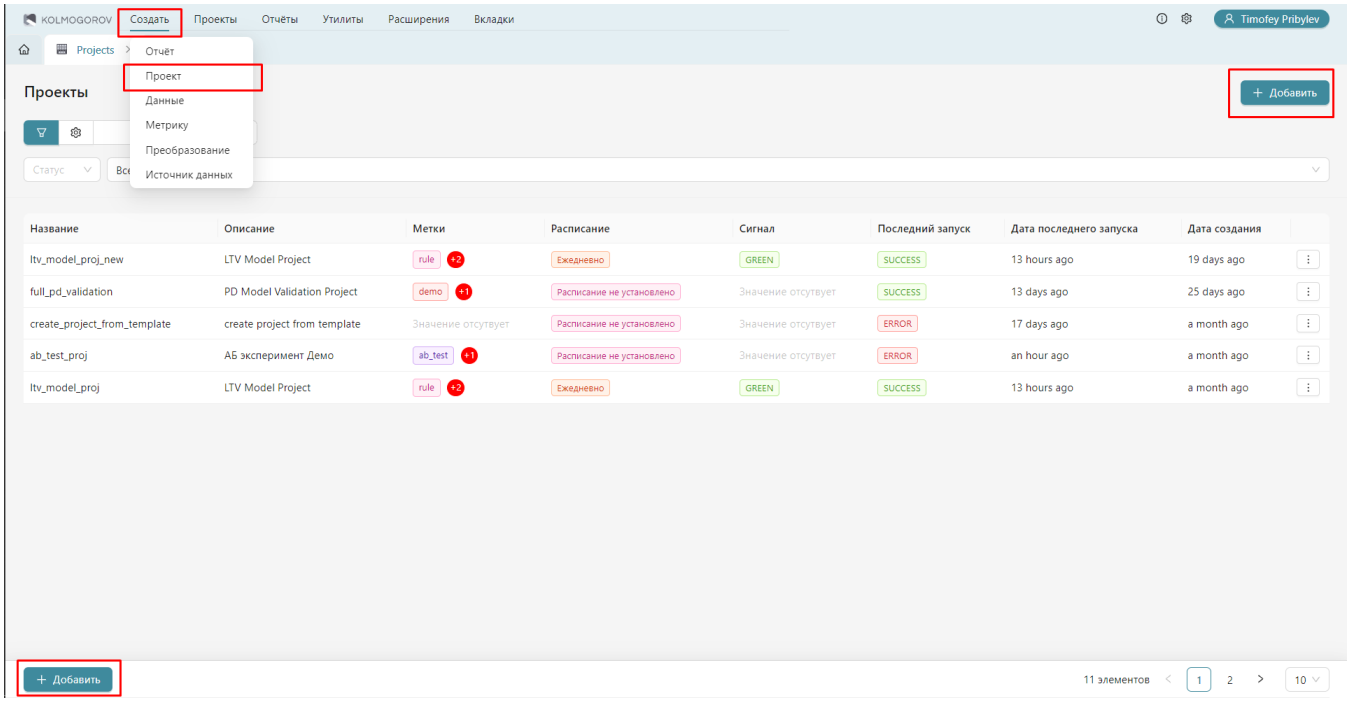
2.5.1 Создание нового проекта

Порядок создания проекта для единоразового или многократного мониторинга работы модели.

Переход к форме создания проекта

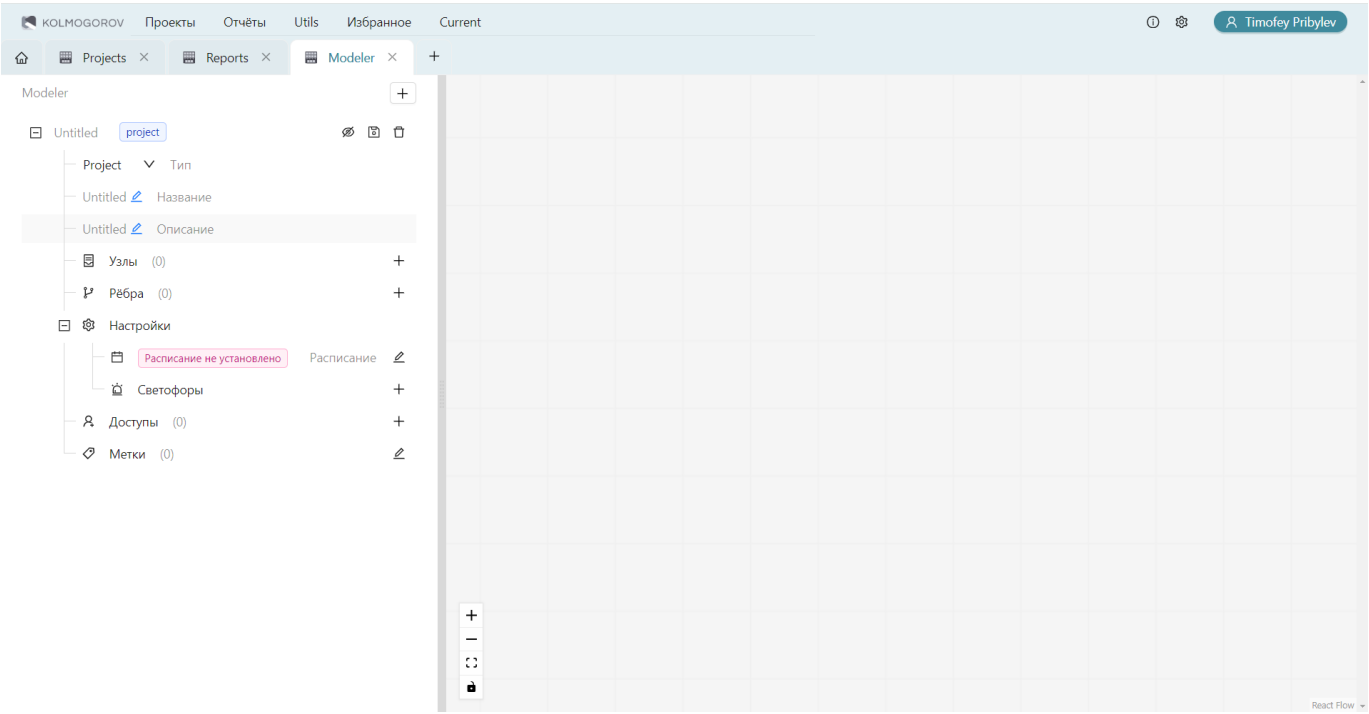
Переход к форме создания нового проекта осуществляется из основного меню приложения одним из следующих способов.

- Способ 1: *Панель управления* > *Создать* > *Проект*
- Способ 2: *Панель управления* > *Проекты* > *Добавить*



- Алгоритм создания проекта из шаблона описан на отдельной странице.

В результате откроется редактор для создания нового проекта:



Обзор редактора проекта

В редакторе проекта есть 8 основных разделов:

1. Выбор типа объекта (проект или шаблон проекта). В данном случае автоматически выбран тип "Проект"
2. Заполнение/редактирование основной информации (уникальное название и описание)
3. Добавление/удаление/редактирование узлов (данных, преобразований или метрик)
4. Добавление/удаление/редактирование ребер (связей между узлами)
5. Настройка расписания
6. Добавление/удаление/редактирование светофоров проекта
7. Добавление/удаление/редактирование доступов
8. Добавление/удаление/редактирование меток

Кроме того, для проекта доступны кнопки просмотра диаграммы с отображением узлов и ребер между ними, сохранения проект и удаления несохраненных данных о проекте.

Также в редакторе есть кнопка добавления нового объекта. С ее помощью можно одновременно создавать несколько проектов.

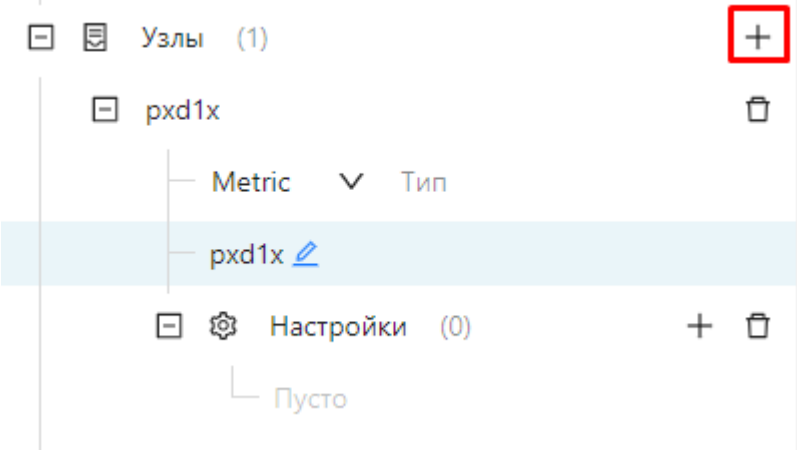
Название и описание проекта являются текстовыми полями, которые вводятся пользователем. Название проекта должно быть уникальным и содержать только латинские буквы, цифры, знаки "дефис" и "нижнее подчеркивание".

После заполнения всех параметров проекта необходимо нажать на кнопку сохранения и в появившемся всплывающем окне подтвердить создание проекта.

Подробнее о других настройках ниже.

Настройка узлов проекта

После добавление узла с помощью кнопки «+» появляется форма для настройки узла:



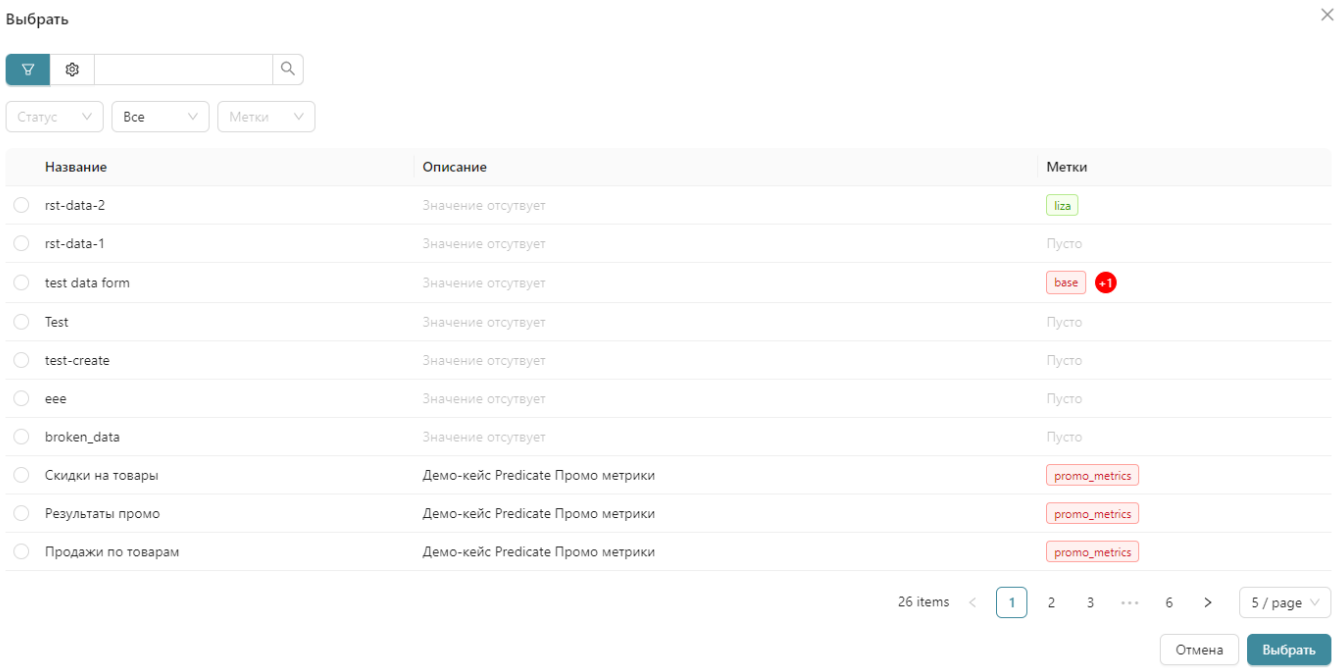
В форме доступны следующие поля:

1. Тип узла: выпадающий список с вариантами «Метрика», «Преобразование» или «Датасет».
2. Алиас узла: техническое название узла, по которому будет производиться обращение к его содержимому в рамках проекта. Например, по алиасу узла с датасетом будет производиться обращение к данным в параметрах метрики. Алиас должен содержать только латинские буквы и цифры, и не должен повторяться в рамках проекта.
3. Настройки: набор дополнительных полей, зависящих о типа узла, с кнопкой «+», предназначенной для добавления в узел конкретного объекта соответствующего типа

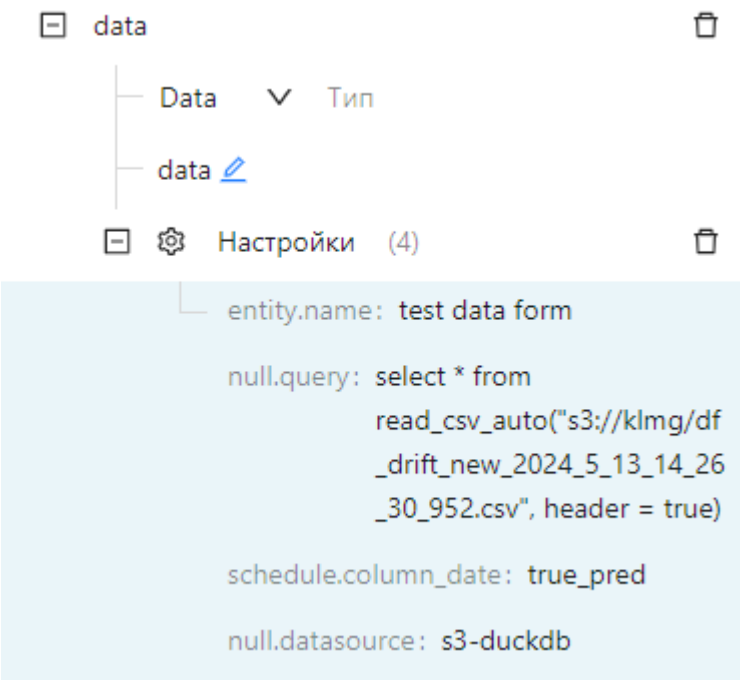
Ниже приведены описания настроек по типам объектов.

УЗЕЛ ДАТАСЕТА

При выборе типа узла «Датасет» кнопка «+» для настроек раскроет модельное окно с каталогом датасетов:

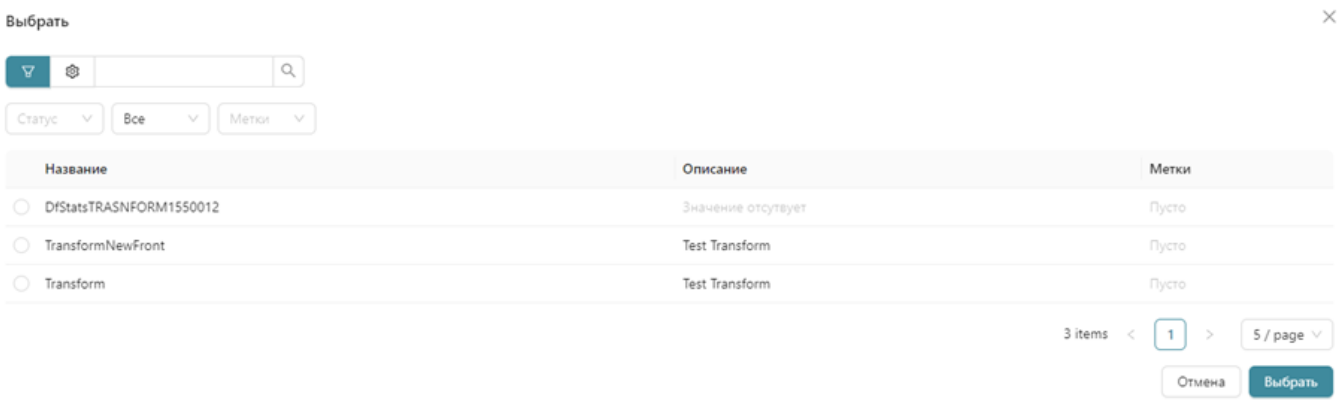


После выбора датасета из каталога в пункте «Настройки» появятся поля с параметрами, описывающими выбранный датасет. Данные поля являются информационными, дальнейшая настройка узла не требуется:

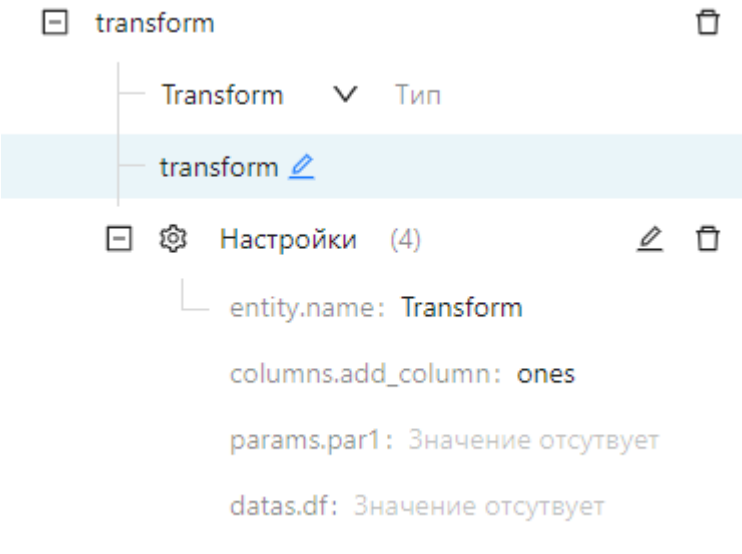


УЗЕЛ ПРЕОБРАЗОВАНИЯ

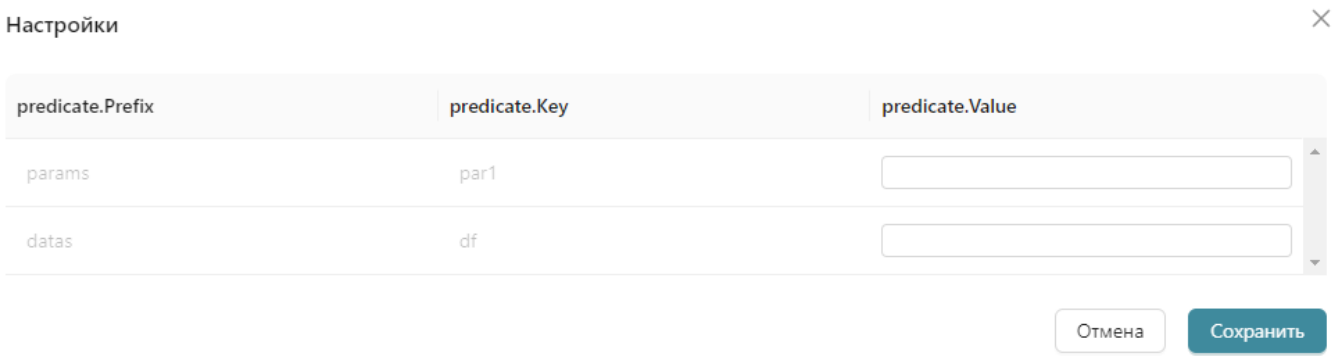
При выборе типа узла «Преобразование» кнопка «+» для настроек раскроет модельное окно с каталогом преобразований:



После выбора преобразования из каталога в пункте «Настройки» появятся поля с параметрами, описывающими выбранное преобразование:

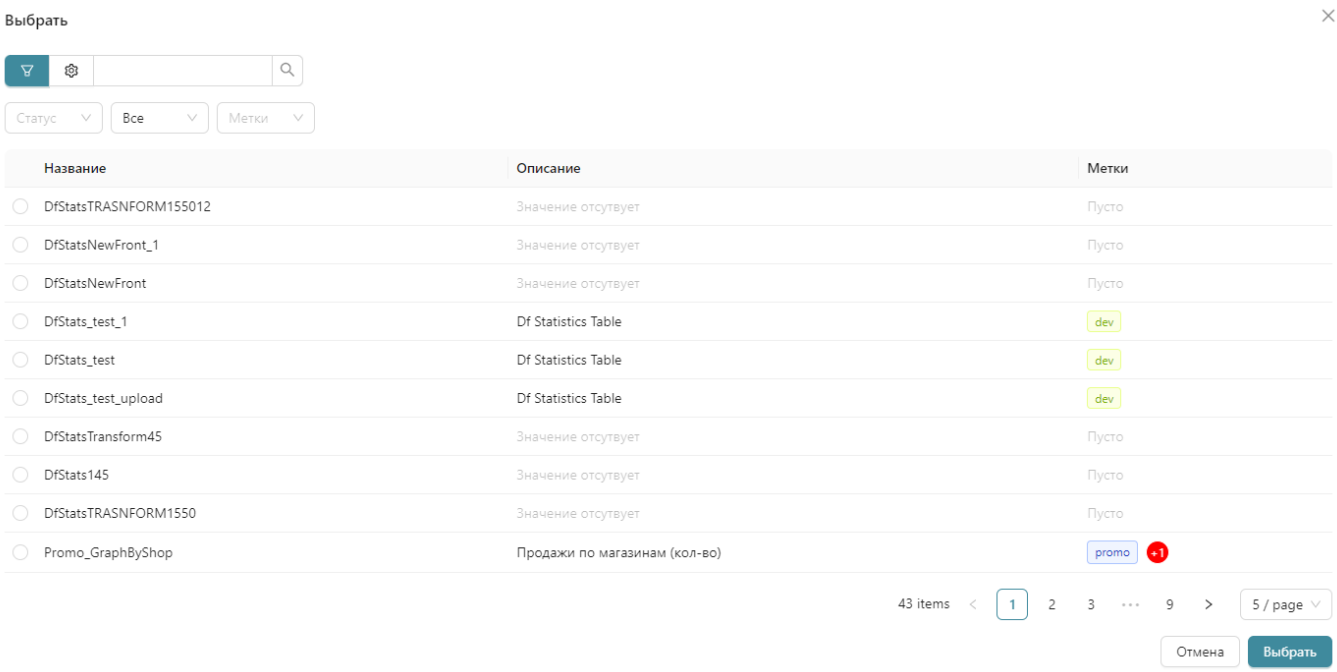


Часть из параметров редактируемы и являются входными данными для преобразования, поэтому необходимо нажать на кнопку редактирования и в открывшемся окне ввести необходимые данные:

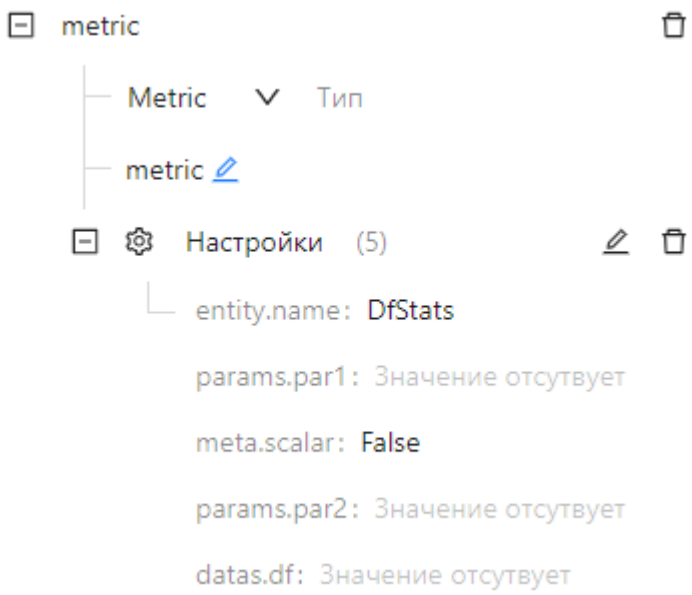


УЗЕЛ МЕТРИКИ

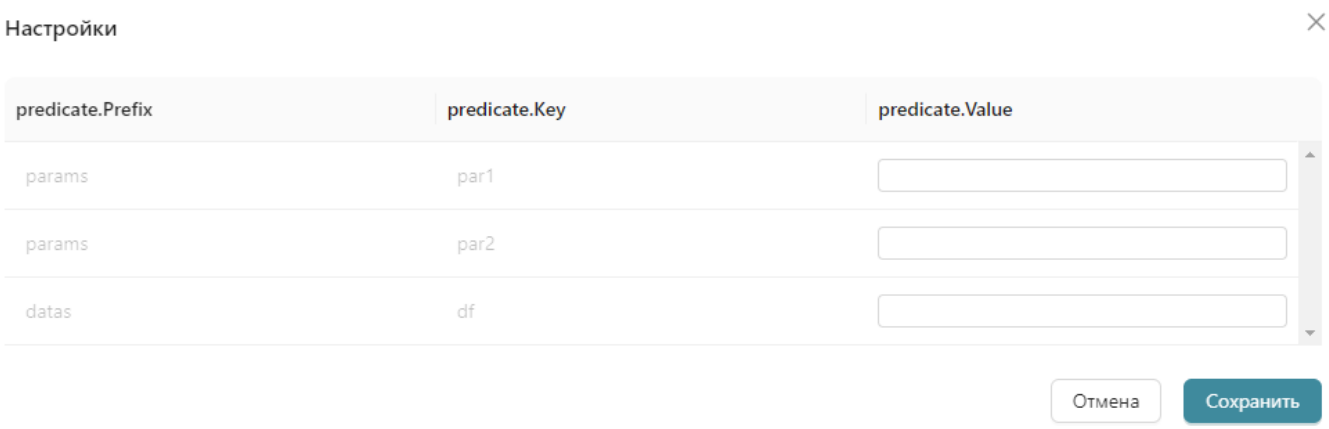
При выборе типа узла «Метрика» кнопка «+» для настроек раскроет модельное окно с каталогом метрик:



После выбора метрики из каталога в пункте «Настройки» появятся поля с параметрами, описывающими выбранную метрику:



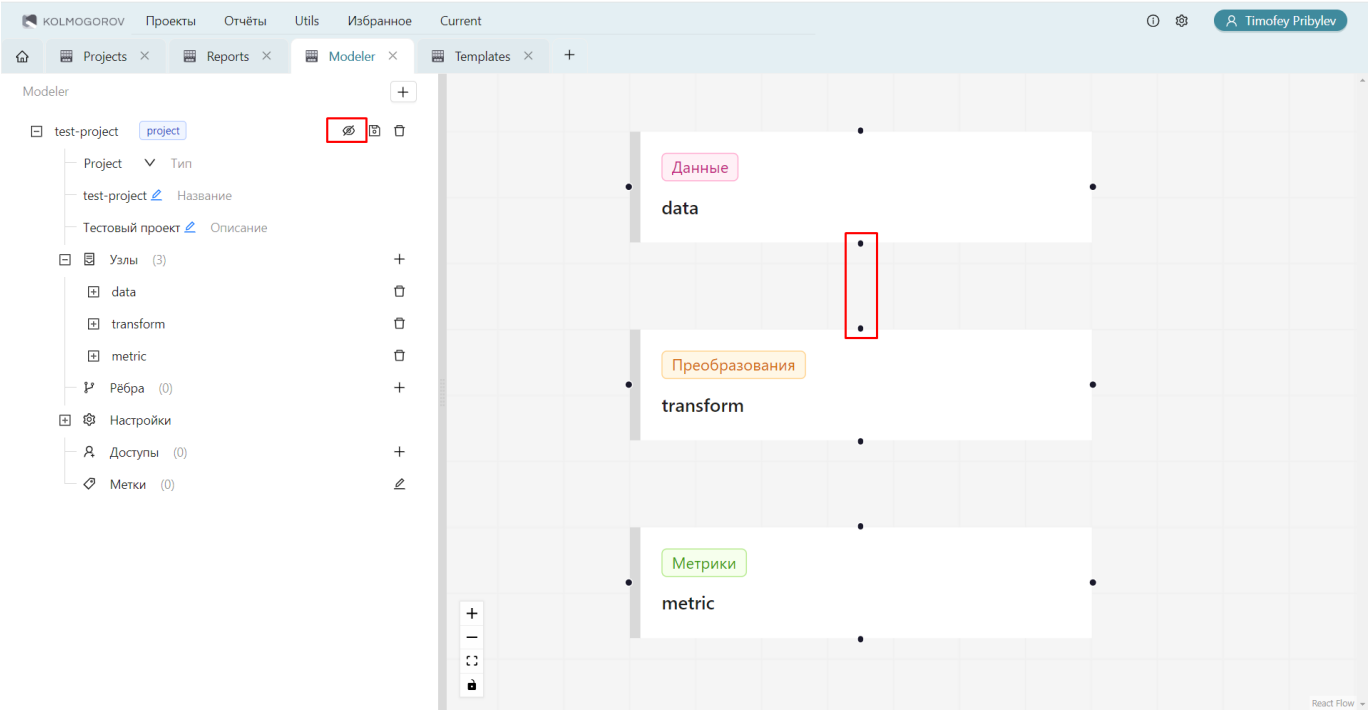
Часть из параметров редактируемы и являются входными данными для метрики, поэтому необходимо нажать на кнопку редактирования и в открывшемся окне ввести необходимые данные:



Настройка ребер проекта

Ребра соединяются узлы друг с другом для определения последовательности выполнения узлов.

Для настройки ребер необходимо нажать на кнопку отображения диаграммы проекта, а затем соединить узлы в нудной последовательности, протянув стрелку от одного узла к другому. Ребра определятся автоматически в соответствующем поле:



Важно учитывать ограничения на работу с ребрами для разных типов узлов:

1. Датасет: нельзя создавать входящие в узел ребра, исходящие ребра могут вести в узел преобразования или метрики.
2. Преобразование: входящие ребра могут исходить из узла датасета или другого преобразования, исходящие ребра могут вести в узел преобразования или метрики.
3. Метрика: входящие ребра могут исходить из узла датасета или преобразования, нельзя создавать исходящие ребра

Ограничений на число входящих и исходящих ребер для любого узла нет.

Дополнительной параметризации у ребер нет.

Настройка расписания

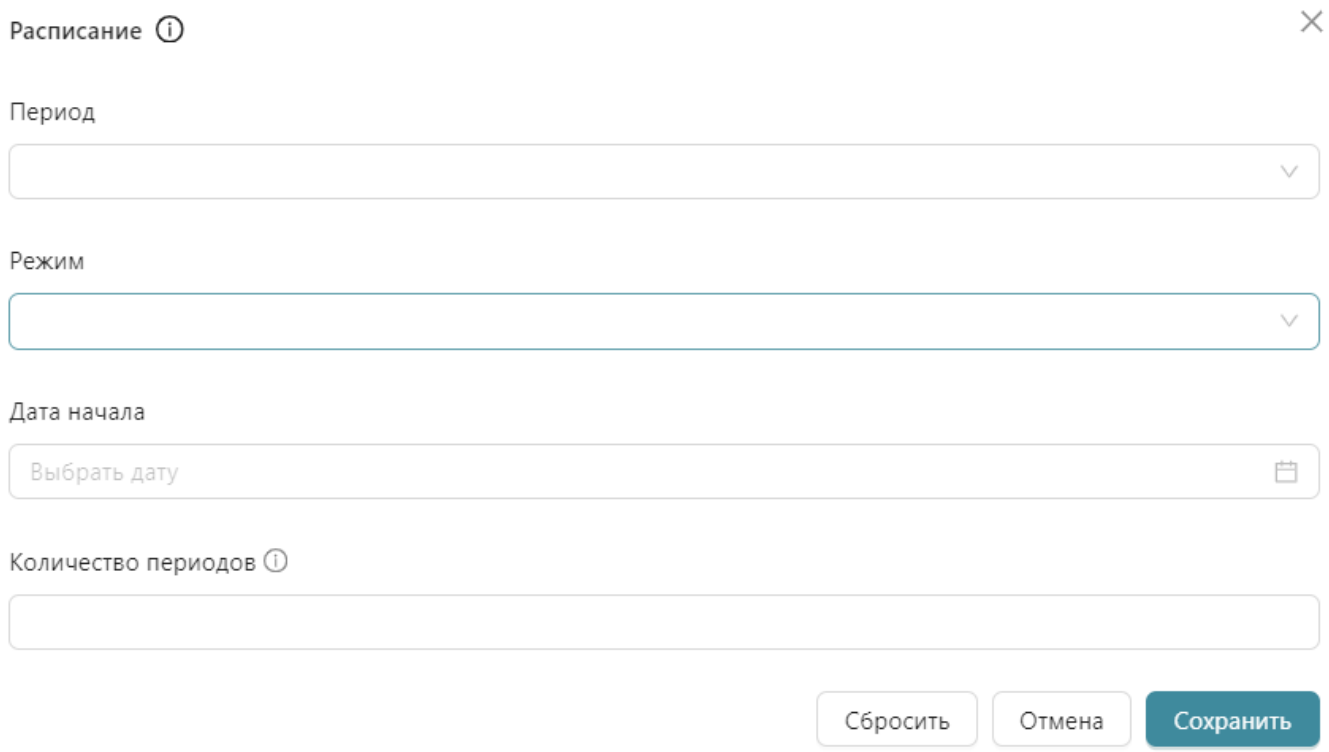
ЕДИНОРАЗОВЫЙ ЗАПУСК

Важно

Если планируется **единоразовый запуск** проекта, не проводите настройку расписания.
Расчет начнется сразу после создания проекта.

РЕГУЛЯРНЫЕ ЗАПУСКИ

После выбора настройки расписание откроется модальное окно настройки расписания:



Период (обязательно) - периодичность запуска проекта:

- Ежедневно;
- Еженедельно;
- Ежемесячно.

Режим (обязательно) - режим забора данных из источника:

- От даты запуска - период отсчитывается от даты запуска;
- Ближайший законченный полный период - ближайший законченный полный период (неделя с пн по вс / месяц с 1 по 31 число / ...).

Дата начала (обязательно) - дата начала регулярного мониторинга. От этой даты "назад" отсчитываются периоды для забора данных.

Количество периодов (обязательно) - количество периодов для забора данных (ширина скользящего окна), данные за это число периодов будут учтены в расчетах.

Пример

Пусть датой начала проекта выбрано 20 октября 2022 (четверг), период - еженедельно, количество периодов - 2.
Тогда в случае выбора режима *От даты запуска* расчеты будут проводиться на данных за 6-20 октября 2022 (2 недели с чт по чт), а при выборе режима *Ближайший законченный полный период* - на данных за 3-17 октября 2022 (2 недели с пн по пн).

Важно

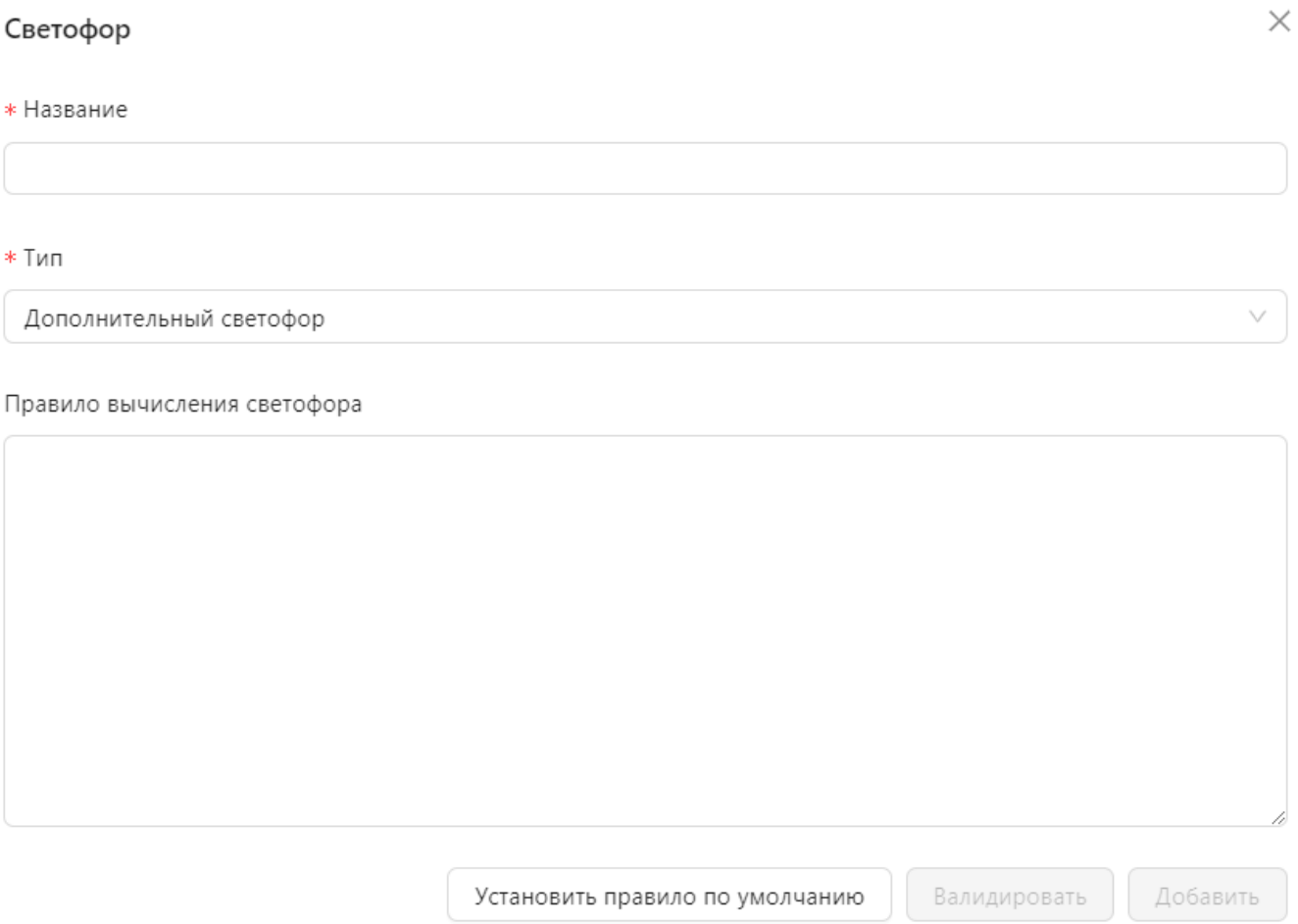
Настройка **скользящего окна** для забора данных при регламентных запусках проекта доступна только для датасетов с отмеченным *Столбцом, содержащим дату наблюдения* при создании датасета.

Для датасетов без такого столбца регламентный запуск проекта также возможен. В этом случае при каждом запуске для расчета берутся **все записи**, имеющиеся в объекте данных на текущий момент.

Настройка светофоров

Светофор проекта – это показатель качества метрик всего проекта в целом. Светофор может быть зеленым, желтым, или красным. Также светофор может быть главным и дополнительным. Главный светофор отображается в каталоге проектов.

При нажатии на кнопку добавления светофора появляется модальное окно настройки светофора:



Параметры светофора следующие:

1. Название (обязательно)
2. Тип (обязательно):

a. Главный светофор - отображается в каталоге как светофор по проекту, может быть не более одного;

b. Дополнительный светофор - отображается только в проекте, может быть сколько угодно.

3. Правило вычисления светофора (обязательно).

Правило светофора можно заполнить автоматически с помощью кнопки «Установить правило по умолчанию». В таком случае создастся правило, которое использует все метрики проекта, которые возвращают скалярное значение. Логика правила будет основываться на светофоры метрик – если все светофоры метрик зеленые – светофор проекта зеленый. Если есть хоть один красный светофор метрики – светофор проекта красный. В иных случаях светофор проекта желтый.

Альтернативный метод создания правила – ручное написание правила, после чего необходимо нажать на кнопку «Валидировать» для проверки правильности написанного правила. При написании правила необходимо придерживаться следующего синтаксиса:

1. Правило представляет собой логическое выражение формата `If условие then действие [elif условие then действие] else действие`. Блок `elif` может повторяться ноль и более раз.
2. Действие – это либо блок с присвоение цвета светофора при выполнении условия. Имеет вид `result = <GREEN|YELLOW|RED>`. Либо вложенное if-выражение той же структуры.
3. Условие – это логическое выражение, содержащее сравнения скалярных значений метрик или их светофоров с какими-либо значениями:

a. Для обращения к скалярному значению метрики необходимо использовать метод `scalar(<алиас ноды с метрикой>)`. Доступные логические операторы: `>`, `<`, `>=`, `<=`, `==`, `!=`. Сравнение производится либо со скалярным значением другой метрики, либо с действительным числом.

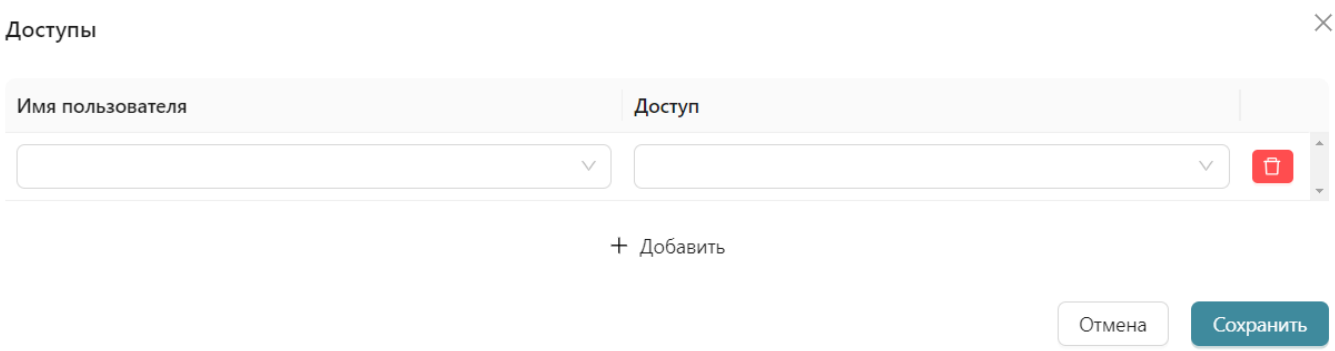
b. Для обращения к светофору метрики необходимо использовать метод `signal(<алиас ноды с метрикой>)`. Доступные логические операторы: `==`, `!=`. Сравнение производится либо со светофором другой метрики, либо с константами `RED`, `YELLOW`, `GREEN`.

c. В логическом выражении также могут использоваться круглые скобки и операторы `OR` и `AND`.

После заполнения всех параметров светофора необходимо нажать кнопку «Добавить».

Настройка доступов

Настройка доступов позволяет определить перечень пользователей, которые могут получить доступ к создаваемому проекту. При нажатии на кнопку настройки доступов открывается соответствующее модельное окно:



На форме доступны следующие поля и кнопки:

1. Добавить – добавляет еще одну строчку для заполнения пары пользователь-уровень доступа.
2. Имя пользователя – выпадающий список из имен пользователей системы.
3. Доступ – уровень доступа, выдаваемый пользователю:

a. Владелец – полный доступ с возможностью редактирования проекта;

b. Только чтение – доступ только на просмотр результатов отработки проекта.

После заполнения всех параметров доступа необходимо нажать кнопку «Сохранить».

Настройка меток

Метки – это теги проектов. При нажатии на кнопку настройки меток появится окно с выпадающим списком доступных меток, где можно либо выбрать существующие метки, либо написать собственную метку. Число меток на одном проекте не ограничено.

Просмотр созданного проекта

После того как проект создан, в каталоге проектов (*Панель управления > Проекты*) появляется соответствующая этому проекту строка и отображается статус данного проекта.

Инструкция по просмотру отчета с результатами проекта представлена в разделе "результаты мониторинга".

2.5.2 Просмотр результатов мониторинга

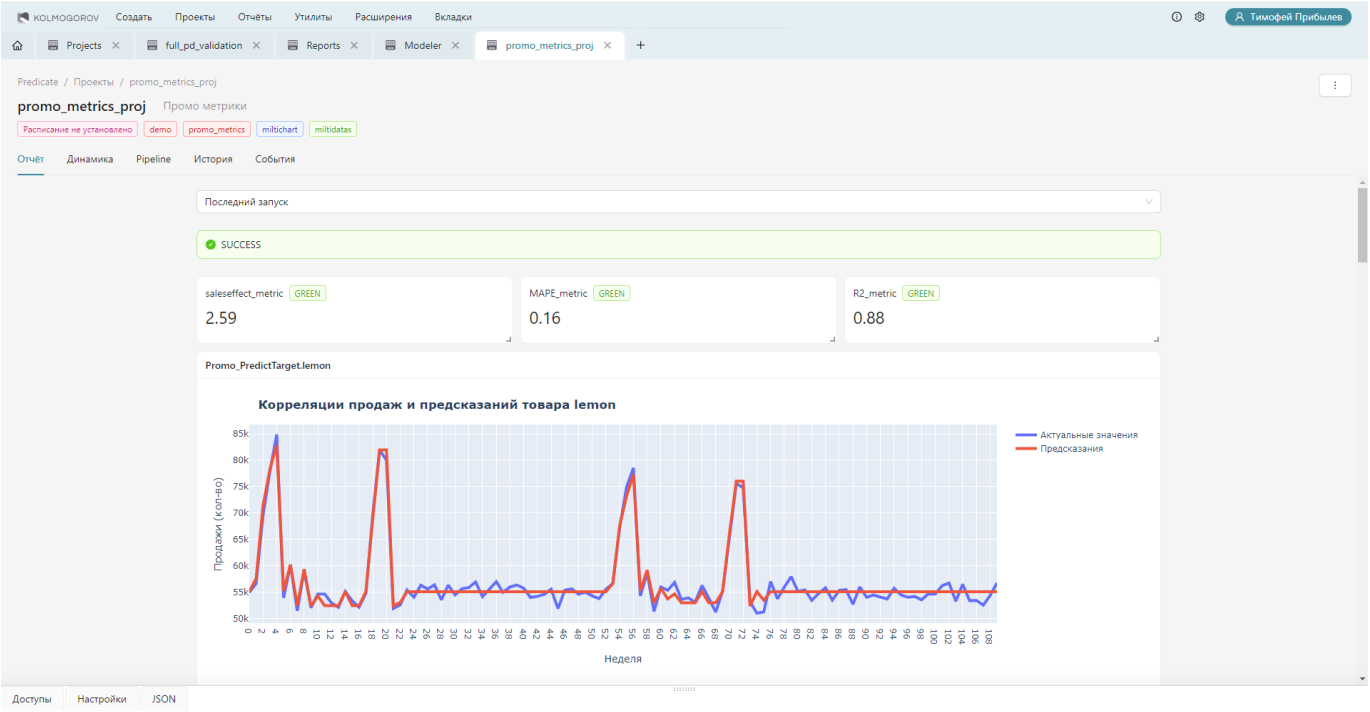
Просмотр дашборда с результатами исполнения проекта.

Переход на страницу проекта

На странице каталога проектов (*Панель управления* > *Проекты*) выберите нужный проект и нажмите на него двойным кликом мыши. Откроется страница проекта в новой вкладке приложения.

Страница проекта

Отчет по проекту представлен ниже:



Кроме отчета по проекту со всеми графиками, скалярными значениями метрик и светофорами проекта, в отчете по проекту доступны также:

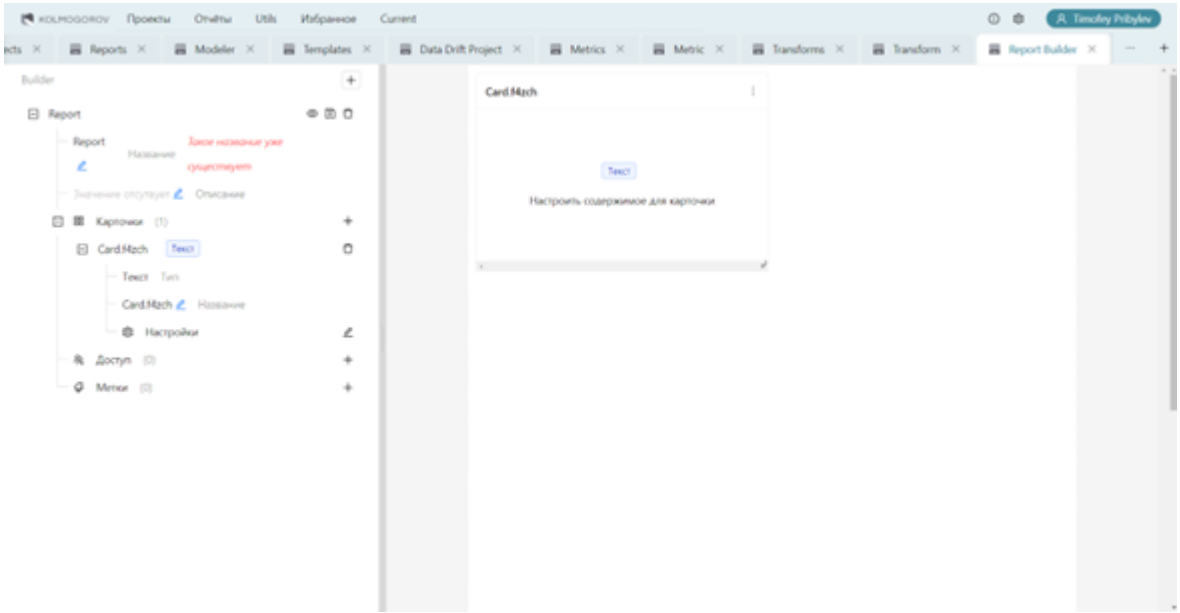
1. Динамика проекта – график с динамикой изменения скалярных значений метрик при множестве исполнений проекта.
2. Pipeline – диаграмма с порядком запуска нод проекта.
3. История – информация о логах запуска нод проекта.
4. События – информация об изменениях, вносимых в проект.
5. Доступы – список пользователей и их доступов к проекту.
6. Настройки – информация о правилах светофоров и о расписании проекта.
7. JSON – техническое представление проект в виде JSON-объекта.

Также в каждом отчете доступен просмотр предыдущих запусков проекта с помощью выпадающего списка запусков проекта.

2.6 Отчёты

2.6.1 Создание отчёта

Для создания нового отчета перейдите по пути *Панель управления* > *Отчеты* в открывшейся странице каталога отчетов нажмите кнопку "Добавить". Откроется форма создания отчета:



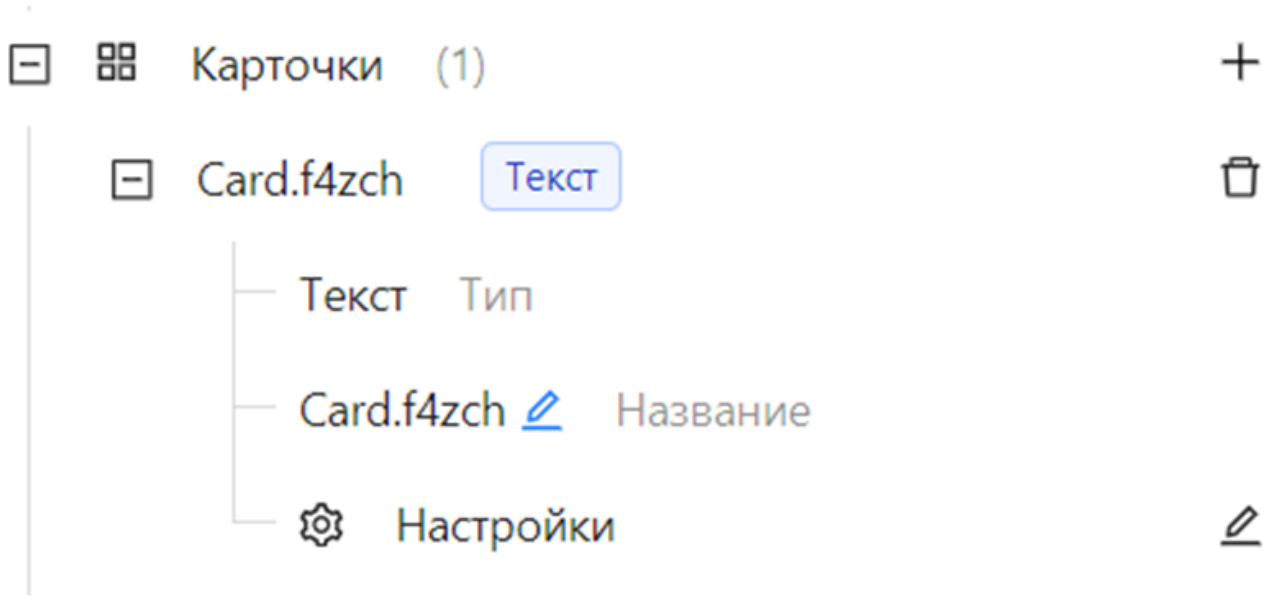
В редакторе отчета есть 4 основных раздела:

1. Заполнение/редактирование основной информации (уникальное название и описание)
2. Добавление/удаление/редактирование карточек с данными отчета.
3. Добавление/удаление/редактирование доступов
4. Добавление/удаление/редактирование меток

Кроме того, доступны кнопки просмотра подготавливаемого отчета, сохранения отчета и удаления несохраненных данных об отчете.

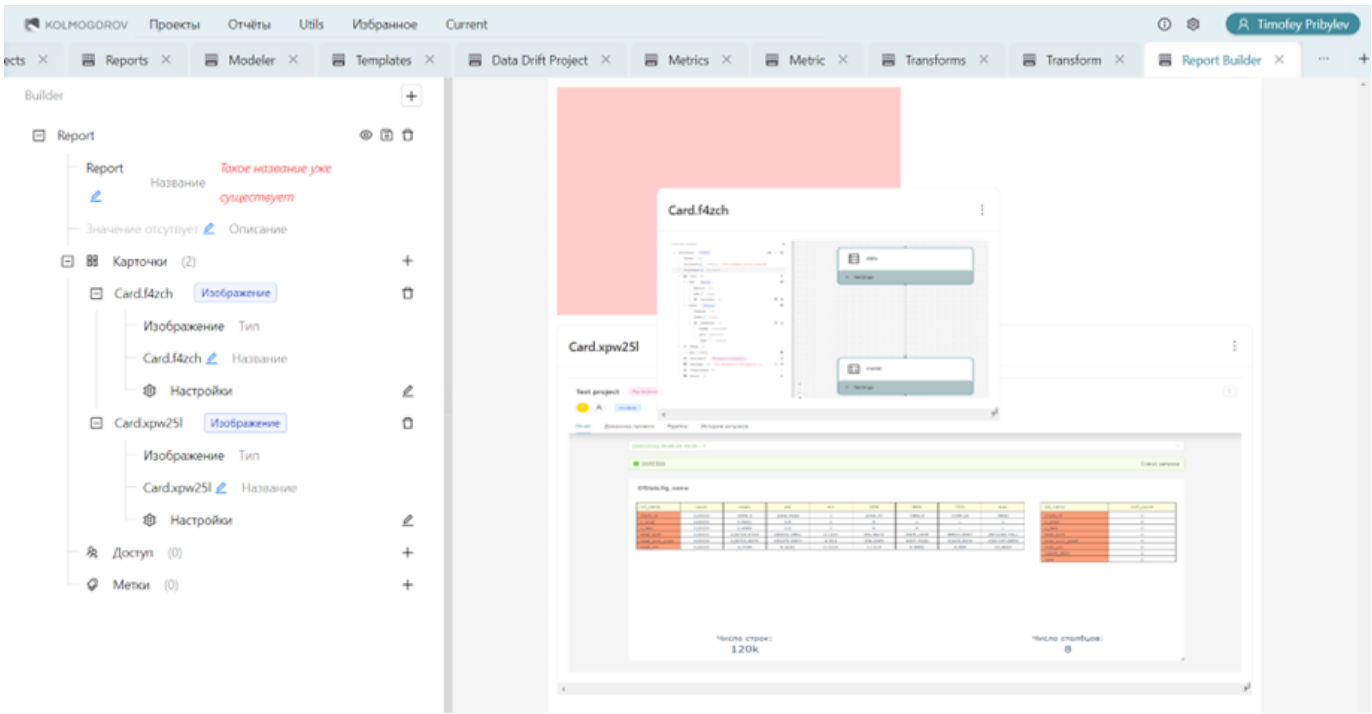
Заполнение основной информации, доступов и метрик аналогично этим же пунктам на форме создания проекта.

При добавлении карточки появляется форма настройки карточки:



Доступные типы карточек: 1. Текстовая карточка. 2. Карточка с пользовательским изображением. 3. Карточка светофора проекта. 4. Карточка таблицы светофоров проекта. 5. Карточка скаляра метрики. 6. Карточка таблицы скаляров проекта. 7. Карточка с графиком метрики. 8. Карточка с динамикой проекта. 9. Карточка таблицы сравнения светофоров разных проектов или запусков. 10. Карточка таблицы сравнения скаляров разных проектов или запусков.

Размеры и относительное положение карточек может быть изменено на поле подготавливаемого отчета:



После заполнения всех параметров отчета необходимо нажать на кнопку сохранения и в появившемся всплывающем окне подтвердить создание отчета.

После создания отчета он доступен в каталоге отчетов для просмотра. Также при просмотре отчета доступа функция скачивания отчета в виде файла формата PDF.