

# ФИНАЛЬНЫЙ ПРОЕКТ

Разработка рекомендательной системы  
для компании-ритейлера

# ЗАДАЧА

Увеличение прибыли от до продаж в интернет-магазине, на 20 %

- РЕШЕНИЕ
- РАЗРАБОТКА РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ
- БИЗНЕС ЦЕЛИ - УВЕЛИЧЕНИЕ ПРИБЫЛИ НА 20 %
- БИЗНЕС МЕТРИКА - PRECISION@3

# ОПИСАНИЕ ДАННЫХ

**events** — датасет с событиями.

- timestamp — время события
- visitorid — идентификатор пользователя
- event — тип события
- itemid — идентификатор объекта
- transactionid — идентификатор транзакции, если она проходила

**category\_tree** — файл с деревом категорий (можно восстановить дерево).

- category\_id — идентификатор категорий
- parent\_id — идентификатор родительской категории

**item\_properties** — файл с свойствами товаров.

- timestamp — момент записи значения свойства
- item\_id — идентификатор объекта
- property — свойство, кажется, они все, кроме категории, захешированы
- value — значение свойства

# ВЫБРАННАЯ МОДЕЛЬ

## LightFM

Это реализация на Python ряда популярных алгоритмов рекомендаций как для неявной, так и для явной обратной связи. Будем использовать LightFM не только из-за его гибридных возможностей, но и из-за того, что он позволяет использовать обычные взаимодействия пользователя с элементом для прогнозирования для известных пользователей. Это очень важно, поскольку у нас не было формальных рейтингов для элементов, мы все равно смогли получить рекомендации на основе этих взаимодействий. Поскольку у нас есть только неявная обратная связь от пользователей, гибкость LightFM в этом отношении полезна.

Для построения рекомендательной системы был использован датасет `events` с выборкой записей по наличию транзакции. В качестве **user** у нас будет выступать **visitorid**, в качестве **item** – **itemid**.

# ПРОВЕДЕННЫЕ ЭКСПЕРИМЕНТЫ

- КОЛОБОРАТИВНАЯ ФИЛЬТРАЦИЯ

- $\text{precision@3} = 0.21$

- XGBOOST

- $\text{precision@3} = 0.0122$

- CatBoostClassifier

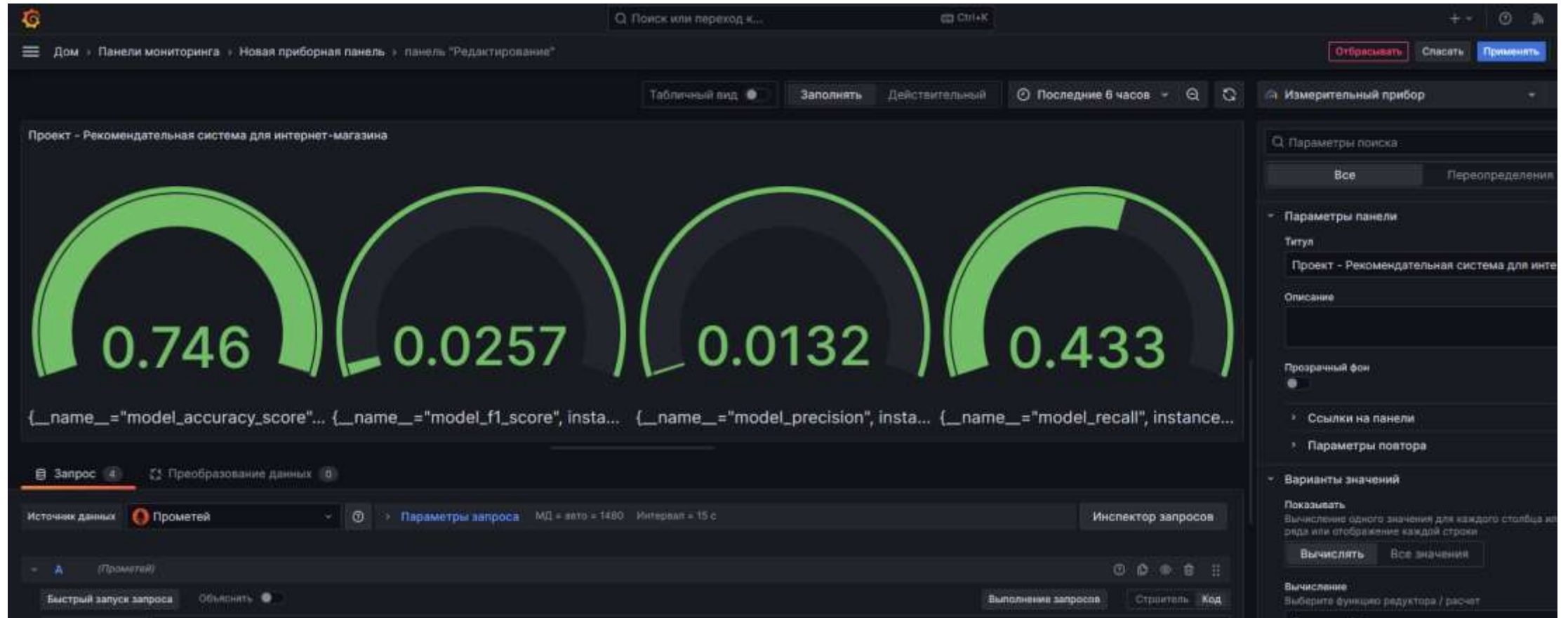
- $\text{precision@3} = 0.0132$

- ALS

- $\text{precision@3} = 0.13$

LightFM = 0.76

# МЕТРИКИ *В Grafana*



# инструмент для сбора метрик Prometheus

The screenshot displays the Prometheus web interface with four query panels. Each panel includes a search bar, a query input field, an 'Execute' button, and a table of results. The panels are for the following queries:

- model\_precision**: Query: `model_precision{instance="localhost:9102", job="windows_exporter"}`. Result: 0.013224962118955006.
- model\_accuracy\_score**: Query: `model_accuracy_score{instance="localhost:9102", job="windows_exporter"}`. Result: 0.745836068370541.
- model\_f1\_score**: Query: `model_f1_score{instance="localhost:9102", job="windows_exporter"}`. Result: 0.82586673207118584.
- model\_recall**: Query: `model_recall{instance="localhost:9102", job="windows_exporter"}`. Result: 0.43540857787818583.

Each panel also features a 'Remove Panel' button and a 'Load time' indicator (e.g., 342ms). The interface includes a top navigation bar with 'Prometheus', 'Alerts', 'Graphs', 'Status', and 'Help' links. A bottom bar contains an 'Add Panel' button.