Центр

Дополнительного

Образования

МГТУ им. Н.Э. Баумана

**ОТЧЕТ**

к итоговой аттестации

курса «Архитектор данных»

ЦДО МГТУ им. Н.Э.Баумана

Выполнил слушатель курса

Круцких О.О.

2023

**Оглавление**

[1. Анализ исходных требований 3](#_Toc150459512)

[1.1. Описание кейса 3](#_Toc150459513)

[2. Техническое проектирование 4](#_Toc150459514)

[2.1. Разработка пайплайна ETL-процесса 4](#_Toc150459515)

[2.2. Обработка файлов 5](#_Toc150459516)

[2.3. Парсинг веб-сайта 7](#_Toc150459517)

[2.4. Структура базы данных 8](#_Toc150459518)

[3. Реализация системы 10](#_Toc150459519)

[3.1. Архитектура 10](#_Toc150459520)

[3.2. Разворачивание системы 11](#_Toc150459521)

[3.3. Пайплайн ETL-процесса 12](#_Toc150459522)

[3.4. Результат ETL-процесса 13](#_Toc150459523)

[3.4.1 Структура данных 13](#_Toc150459524)

[3.4.2 Визуализация 14](#_Toc150459525)

[3.5. Обучение ML-модели 15](#_Toc150459526)

[4. Документация 20](#_Toc150459527)

[5. Выводы 21](#_Toc150459528)

[Приложение 22](#_Toc150459529)

1. **Анализ исходных требований**
   1. **Описание кейса**

В данной работе рассматривается кейс «Исследование продаж велосипедов и сопутствующего снаряжения в зависимости от района проживания клиентов».

Задача актуальна, т.к. позволит торговой сети распределять товар по магазинам и складам оптимально с точки зрения минимизации затрат на перемещение товаров (т.е. логистику).

Согласно задаче, есть два основных источника данных:

* Файлы с данными
* Веб-сайт магазина

Сбор данных и обработку данных необходимо автоматизировать для дальнейшего использования в системах визуализации и машинного обучения.

Следовательно, получаем следующие требования к системе.

Функциональные требования:

* Сбор и хранение информации.
* Интеграция с внешними источниками данных.
* Анализ собранной информации.
* Формирование и поддержка базы знаний.
* Предоставление доступа к базе знаний через пользовательский интерфейс.

Нефункциональные требования:

* Система должна быть масштабируемой и способной обрабатывать большие объемы данных.
* Система должна иметь высокую производительность и отзывчивость.
* Система должна быть надежной и устойчивой к отказам.
* Система должна быть легко интегрируемой с другими системами.
* Система должна быть легко обновляемой и расширяемой.

1. **Техническое проектирование**
   1. **Разработка пайплайна ETL-процесса**

ETL-процесс включает в себя следующие этапы:

* Загрузка исходных данных в базу и создание стейджингового слоя
* Обработка данных и сохранение полученного результата в базу с префиксом dwh
* Выгрузить данные, с помощью парсинга страниц сайта и записать их в базу данных.

Также после обработки, согласно заданию, данные необходимо отобразить в системе визуализации, но, поскольку интеграции с такой системой не требуется, то можно использовать файл для передачи данных и систему визуализации, поддерживающую чтение из файлов.

На рис.1 продемонстрированы все этапы ETL-процесса.

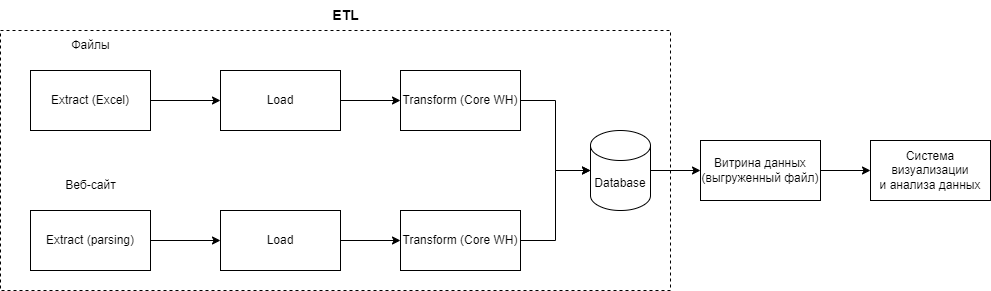


Рис.1 Схема пайплайна ETL-процесса

Далее рассмотрим каждый этап подробнее.

* 1. **Обработка файлов**

Диаграмма BPMN процесса обработки данных представлена на рис.2

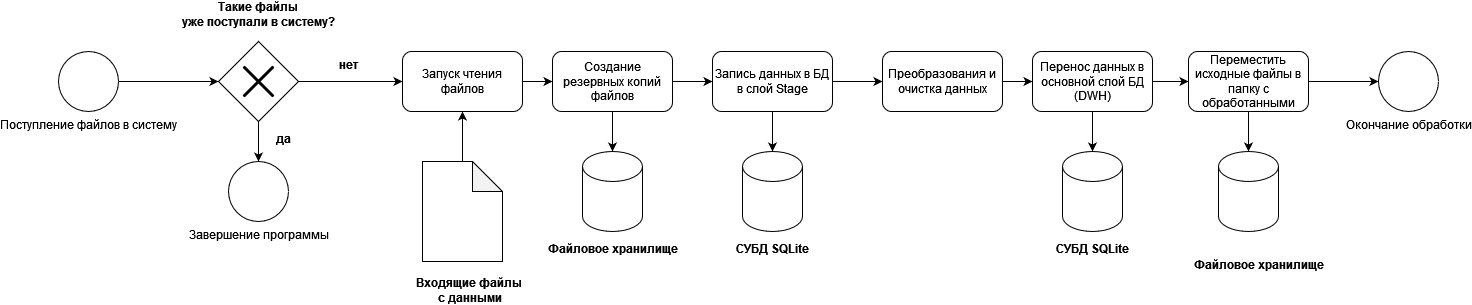


Рис.2 BPMN-диаграмма процесса обработки данных

Данные поступают в виде файлов.

Файл transactions (информация о платежных транзакциях и товарах) содержит следующие столбцы:

transaction\_id

product\_id

customer\_id

transaction\_date

online\_order

order\_status

brand

product\_line

product\_class

product\_size

list\_price

standard\_cost

product\_first\_sold\_date

Файл CustomerDemographic (информация о покупателях) включает столбцы:

customer\_id

first\_name

last\_name

gender

past\_3\_years\_bike\_related\_purchases

DOB

job\_title

job\_industry\_category

wealth\_segment

deceased\_indicator

owns\_car

tenure

Файл CustomerAddress содержит адреса клиентов и представлен структурой столбцов:

customer\_id

address

postcode

state

country

property\_valuation

Файл NewCustomerList содержит список новых клиентов. Его структура:

first\_name

last\_name

gender

past\_3\_years\_bike\_related\_purchases

DOB

job\_title

job\_industry\_category

wealth\_segment

deceased\_indicator

owns\_car

tenure

address

postcode

state

country

property\_valuation

* 1. **Парсинг веб-сайта**

Используемый в данной работе веб-сайт предоставляет следующие данные:

Name – название товара,

type – тип товара,

count – количестов товара в наличии,

count\_measure – единицы измерения количества товара,

source\_id – идентификатор товара в системе веб-сайта,

price\_current – текущая цена товара,

price\_current\_currency –валюта, в которой указана текущая цена товара,

price\_old – предыдущая цена товара (до установления скидки),

price\_old\_currency валюта, в которой указана предыдущая цена товара,

url\_detail – часть url-пути к странице товара.

Также дополняем эти поля данными, которые необходимы для их идентификации в базе данных DWH:

source - адрес веб-сайта,

category – категория товара (как часть пути url-адреса к странице каталога товаров),

webfilter - категория товара (как часть пути url-адреса к странице каталога товаров)

load\_date – дата извлечения данных парсингом.

* 1. **Структура базы данных**

Для хранения данных, источником которых являются файлы, предлагается структура данных, описанная ER-диаграммой на рис.3 Таблица Products содержит информацию о приобретаемых товарах, Transactions – информация о транзакциях покупок, Customers – информация о клиентах, CustomerAddresses – информация об адресах клиентов (проживания или доставки). Данные о новых клиентах из файла NewCustomerList необходимо загружать в таблицу Customers, но поскольку во входящих данных этим новых клиентам не присвоен идентификатор (customer\_id), то в данный момент их загрузка невозможна, и требуется уточнение.

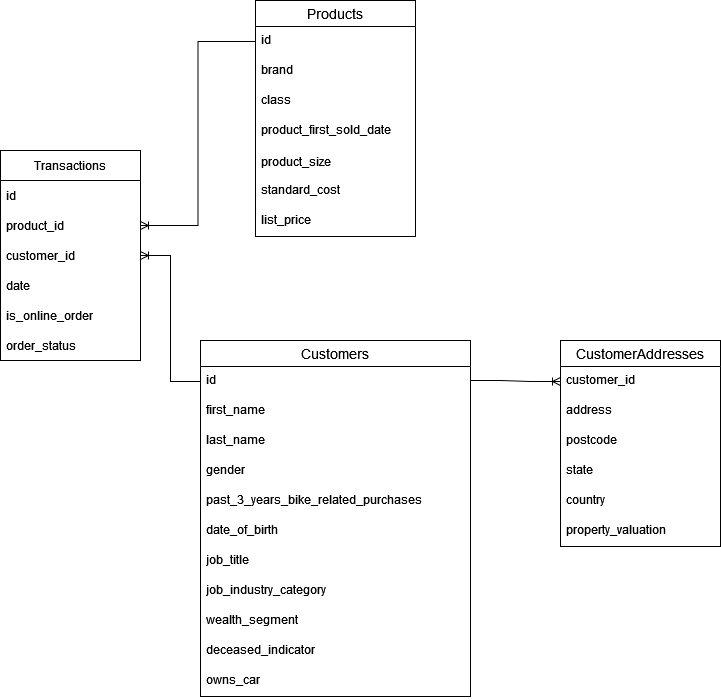


Рис.3 ER-диаграмма структуры базы данных проектируемой системы

Для хранения данных, поступающих в результате парсинга сайта, создадим еще одну таблицу и добавим ее в базу данных. Получим структуру, указанную на рис.4.

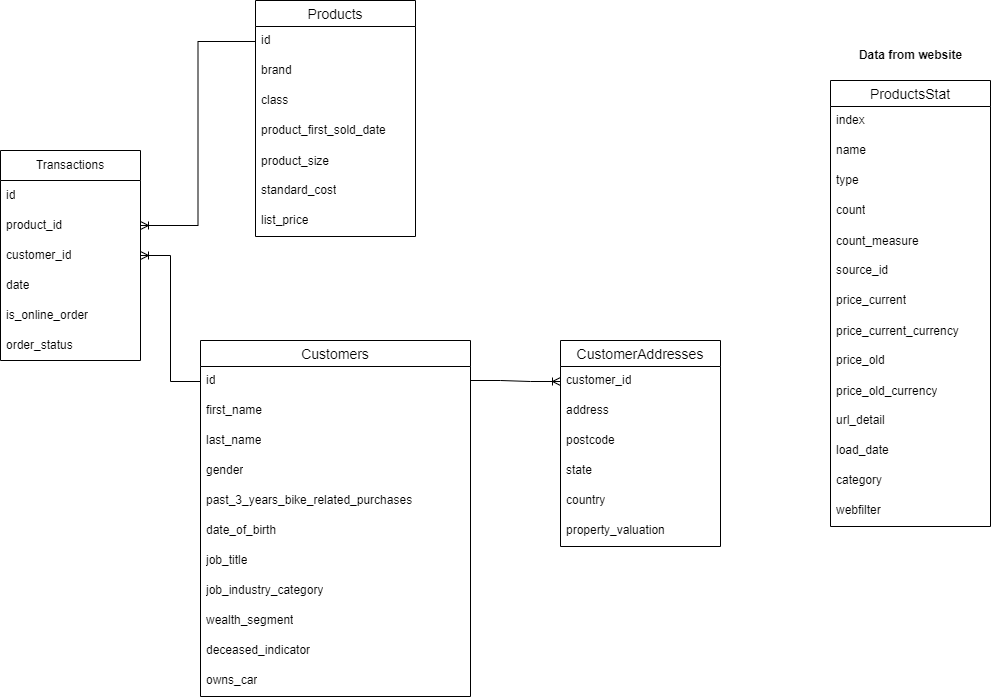


Рис.4 ER-диаграмма структуры базы данных проектируемой системы с таблицей для парсинга

Для связи таблицы с результатами парсинга можно создать таблицу соответствия между идентификаторами сущностей данных из файлов и идентификаторами, используемыми на сайте. Данное решение представлено на рис.5.

Поскольку в текущем случае данные разрозненные, то необходимо провести дополнительную работу по ручному сопоставлению данных из файлов и парсинга. Однако, в задании не указано это требование, и поэтому для разрабатываемой системы достаточно базы данных, структура которой указана на рис. 4.

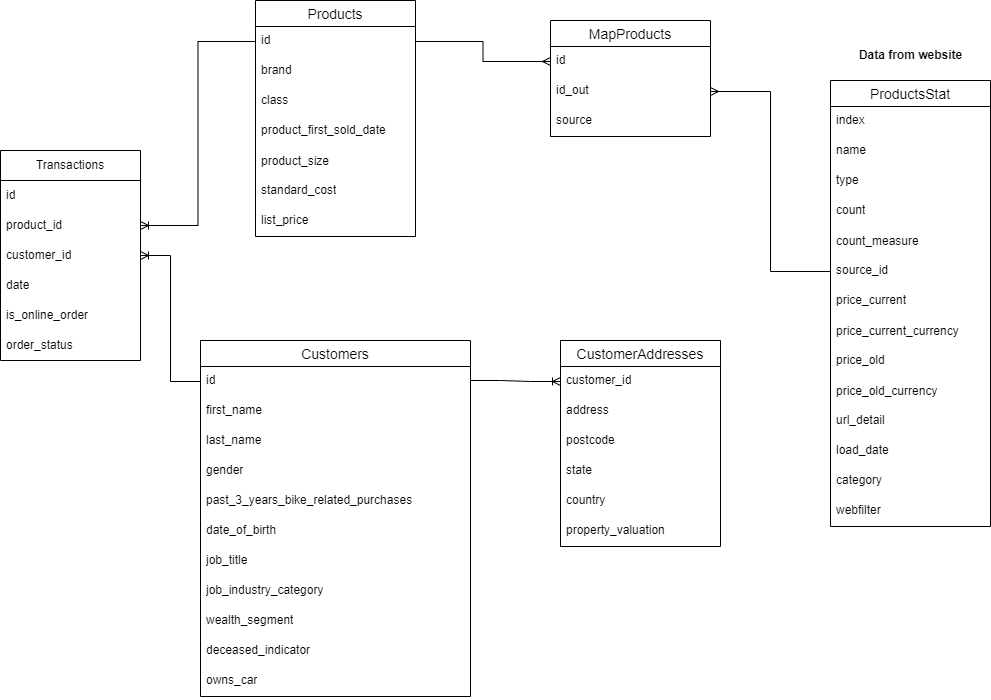


Рис.5 ER-диаграмма структуры базы данных проектируемой системы с таблицей для парсинга и таблицей сопоставления данных

1. **Реализация системы**
   1. **Архитектура**

Предлагается следующая архитектура разрабатываемой системы:

* База данных.
* Скрипты для загрузки данных из входящих файлов в базу данных.
* Скрипт для парсинга веб-сайта.
* Средства автоматизации запуска скриптов.
* Средства виртуализации для разворачивания системы.

В качестве базы данных используется легковесную СУБД sqlite. В дальнейшем предполагается переход на более совершенные системы, поддерживающие транзакции (postgres).

В качестве языка программирования для скриптов загрузки данных и парсинга выбран язык python, т.к. благодаря множеству библиотек он предоставляет решение всех необходимых задач.

Для автоматизации ETL-процесса предлагается Apache Airflow. Данная система позволяет все этапы ETL-процесса представить в виде DAG-файла.

Для автоматизации обучения ML-модели, проведения тестов, просмотра их результатов и метрик используется инструмент MLflow. Для хранения ML-модели предлагается объектное хранидище MinIO (S3).

Для виртуализации рекомендуется Docker как наиболее универсальное средство по развертыванию систем.

* 1. **Разворачивание системы**

Для виртуализации используется Docker как наиболее универсальное средство по развертыванию систем. Используем среду для разворачивания apache/airflow:2.7.1, которую указываем в Docker-файле.

Структура развернутых контейнеров в Docker готового проекта указана на рис.6

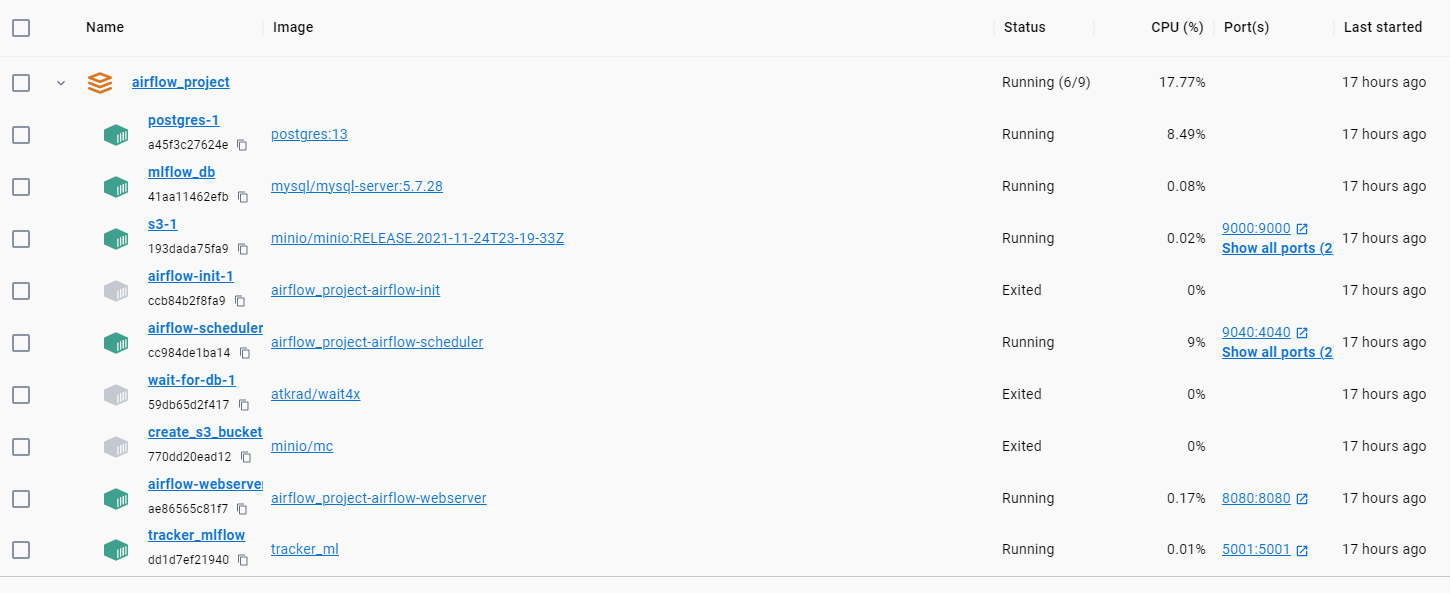


Рис.6 Структура проекта автоматизированного ETL-процесса с использованием Airflow и Docker

* 1. **Пайплайн ETL-процесса**

Система Apache Airflow позволяет автоматизировать ETL-процессы. Для автоматизации создается файл DAG, который содержит в себе список задач в виде функций Python с декоратором @task.

Реализация процесса ETL для файлов данных описана следующими задачами:

* extract\_data\_from\_files\_to\_stage (перенос данных из файлов в стейджинговый слой с минимальными преобразования)
* transfer\_filesdata\_from\_stage\_to\_dwh (перенос данных из стейджингового слоя в DWH со всеми необходимыми преобразованиями)

Процесс ETL для парсинга также разбита на две задачи:

* extract\_web\_data (парсинг сайта и перенос данных в стейджинговые таблицы)
* create\_dwh\_parcer\_data (объединение данных стейджинговых таблиц и перенос в таблицу DWH\_ProductsStat)

Также файл DAG включает в себя задачу create\_outputfile\_for\_dashboards, которая формирует файл “ProductsStat.csv” с данными парсинга сайта, помещенных в DWH. Этот файл используется в качестве источника для построения дашбордов. Он находится в папке output\_data проекта.

На рис. 7 представлена полная структура файла DAG, включающего вышеописанные задачи. Листинг приведен в приложении 1.

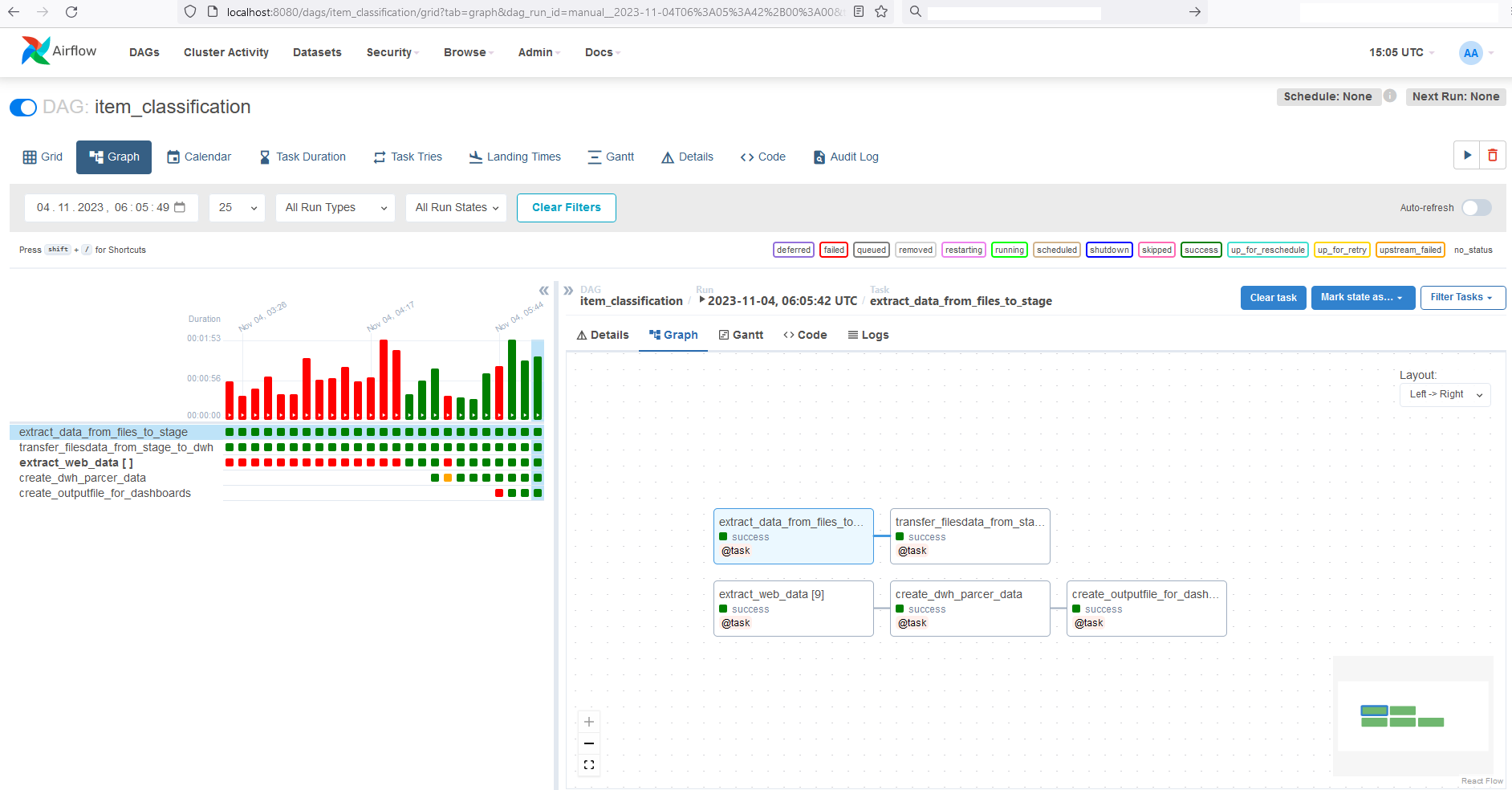


Рис.7 Визуализация графа заданий ETL-процесса DAG-файла

* 1. **Результат ETL-процесса**
     1. **Структура данных**

В результате выполнения ETL-процесса получаем заполненную базу данных DWH (рис.8) данными из файлов и сайта, а также файл “ProductsStat.csv” (рис.9) для построения дашборда.

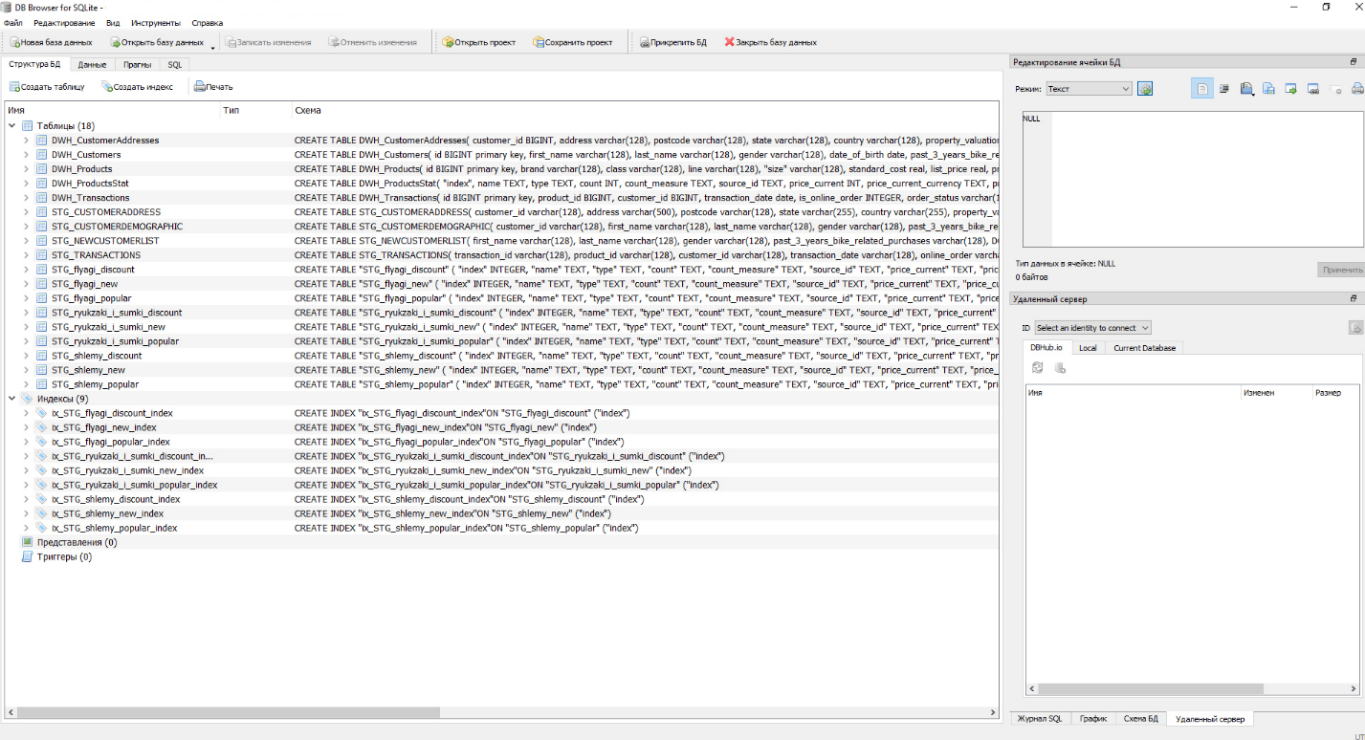


Рис.8. Структура база данных после завершения ETL-процесса

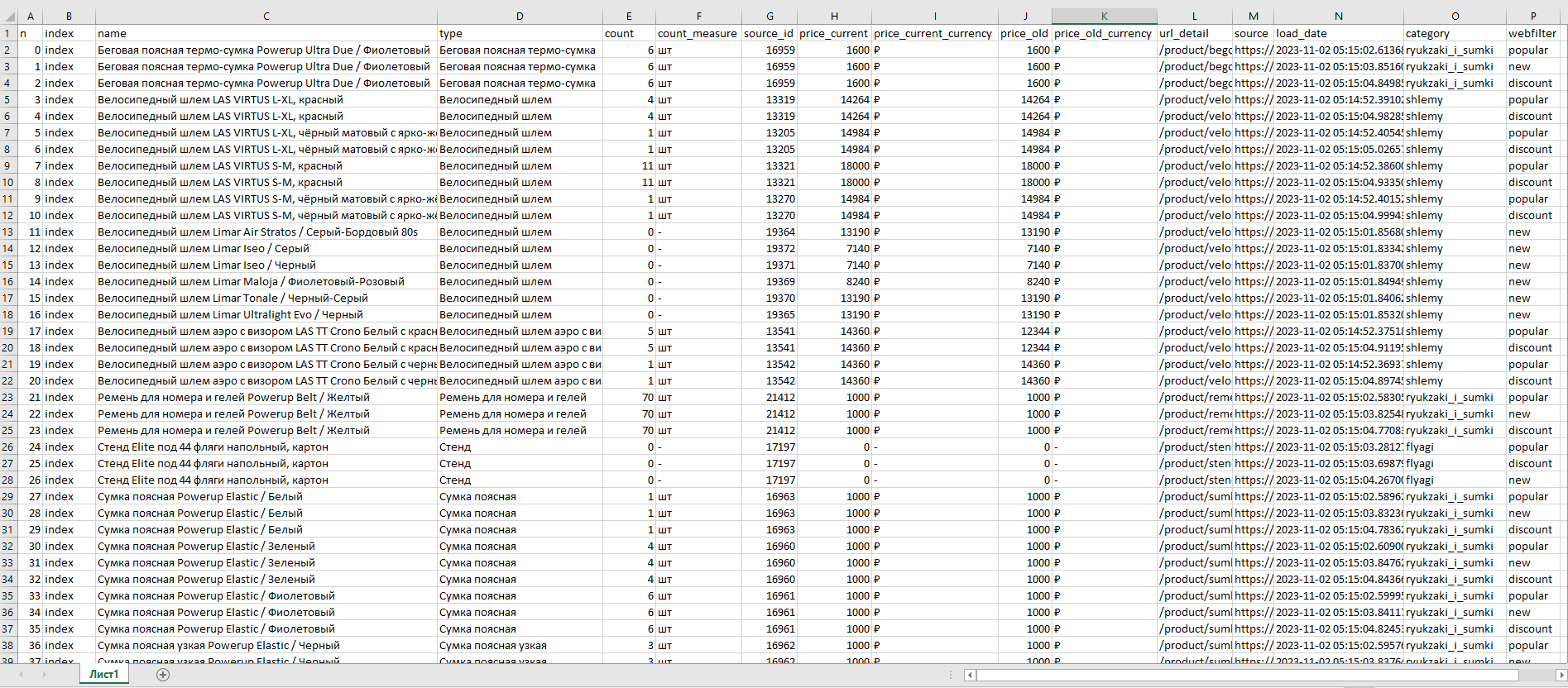


Рис.9 Данные в сформированном файле “ProductsStat.csv”

* + 1. **Визуализация**

Для визуализации используется система Yandex Datalens. Она поддерживает файлы “.csv” в качестве источника данных.

На основе файла “ProductsStat.csv”, который был получен в результате выполнения ETL-процесса, были сформированы следующие чарты:

* Количество товара (файл содержит поле “count” с информацией о количестве товара в наличии)
* Размер скидки (файл содержит данные о текущей цене “price\_current” и предыдущей “price\_old”, что позволяет вычислить размер скидки)
* Максимальная цена (вычисляется на основе поле “price\_current”, используется агрегатная функция “max”)
* Соотношение ассортимента (вычисляется на основе поля “count ”)

На каждом чарте можно видеть несколько данных – разбиение товаров осуществлено по категориям.

Также дашборд обладает тремя фильтрами, значения которых можно применить к графикам:

* Категория товара.
* Тип сортировки.
* Временной период.

На рис. 10 представлена визуализация данных в Yandex DataLens

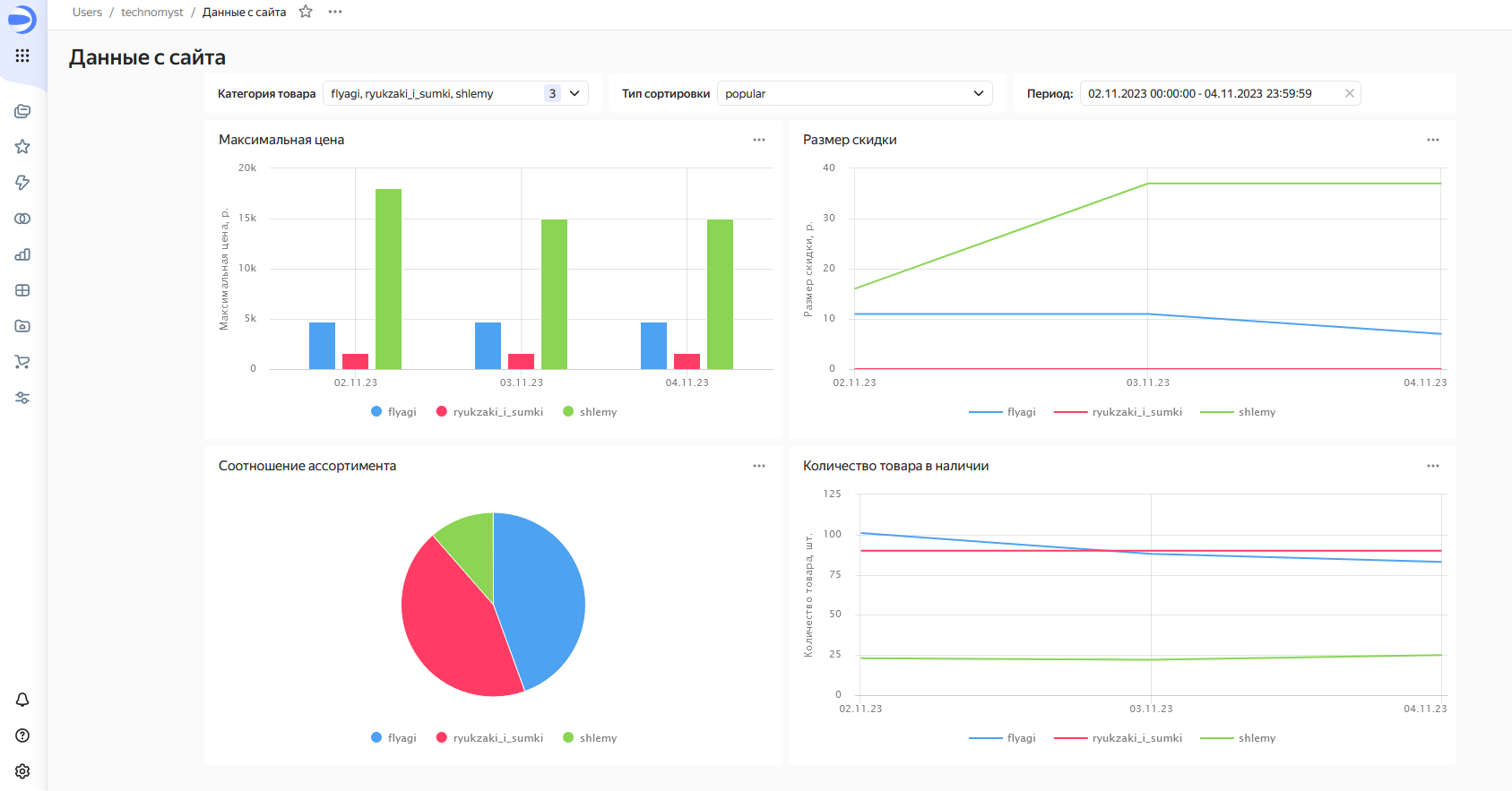


Рис. 10 Визуализация данных в Yandex DataLens

* 1. **Обучение ML-модели**

Для автоматизации обучения ML-модели используется инструмент MLflow, а для хранения - MinIO. Разворачиваются данные системы также в Docker-контейнерах

В данном случае пайплайн работы с ML-моделью включает:

* Подготовка данных
* Разбиение данных на обучающую и тестовую выборки
* Обучение модели
* Получение метрик и оценка модели
* Тестирование/использование на реальных продуктовых данных

Весь цикл работы с ML-моделью также описывается в файле DAG системы Airflow. Схема задач показана на рис.11

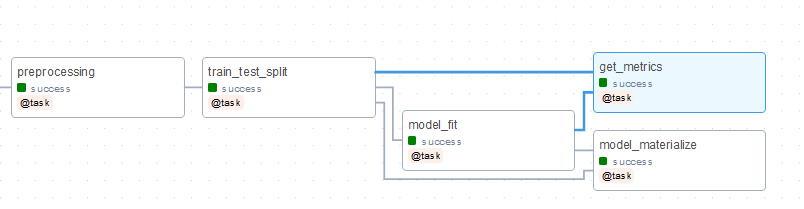


Рис. 11 Структура пайплайна работы с ML-моделью в файле DAG

Полная структура файла DAG, включающего ETL-процесс и работу с ML-моделью, показана на рис.12.

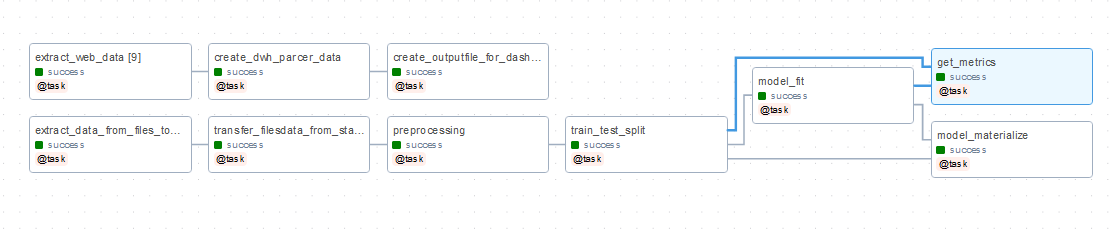


Рис. 12 Полная структура всего пайплайна задания, включающего ETL-процесс и работу с ML-моделью, в файле DAG

Также базу данных пополнят четыре таблицы (рис.13), предназначенные для обучения модели (model\_data,train\_data,test\_data) и хранения результатов ее предсказания (model\_predict).

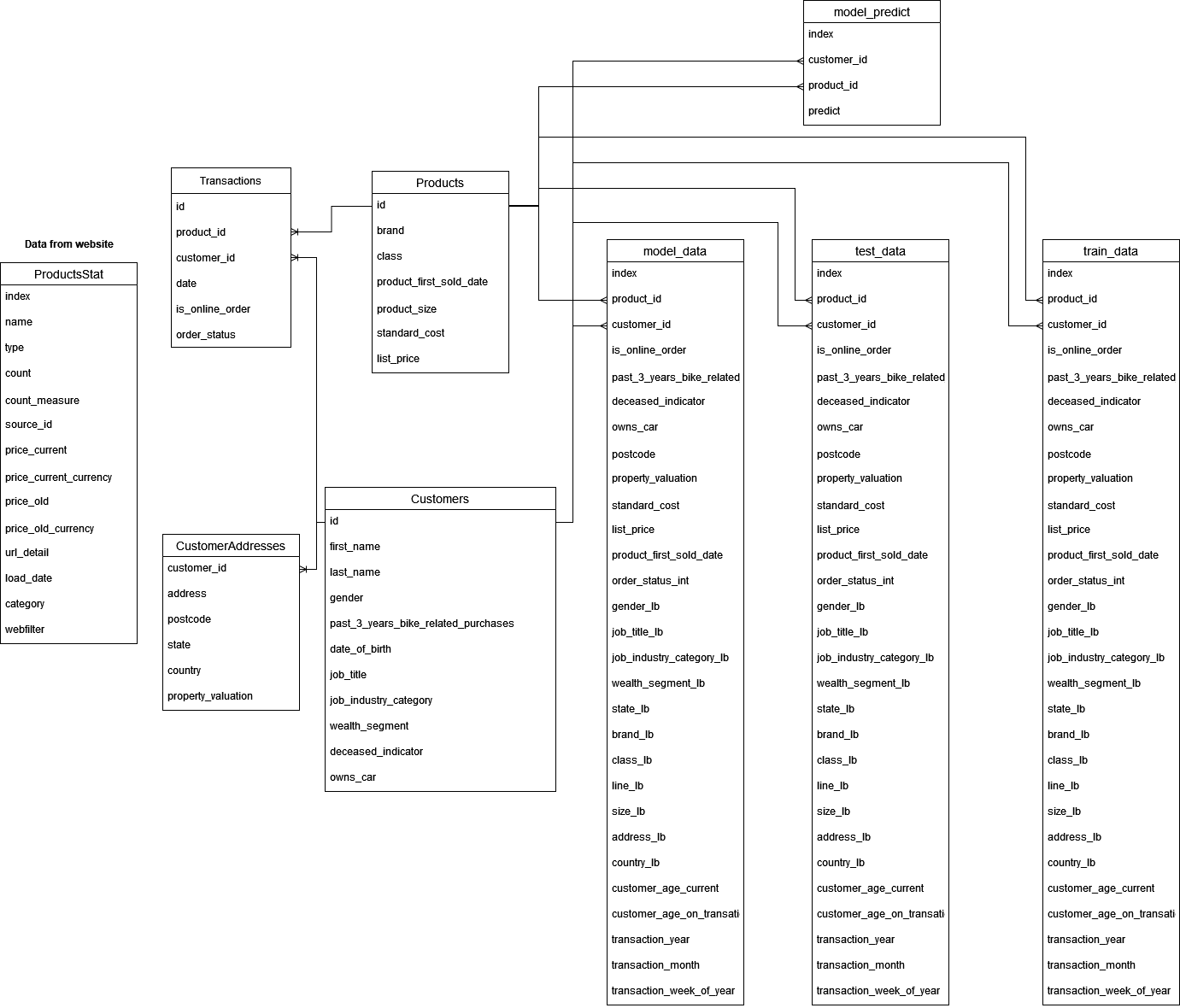


Рис. 13 Полная структура базы данных, включающая таблицы для обучения ML-модели

При успешном выполнении пайплайна, описанного в файле DAG, в графическом интерфейсе MLflow отобразится обученная модель (рис.14), а также список проведенных тестов (рис.15) и полученных метрик (рис.16). В таблице model\_predict появятся данные, предсказанные обученной ML-моделью.

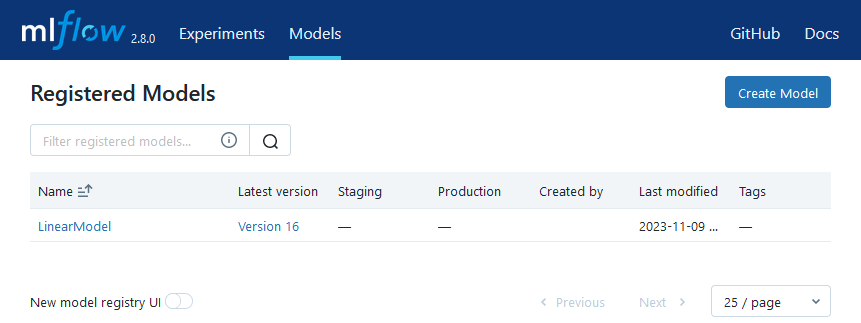


Рис.14 Отображение модели в интерфейсе MLflow

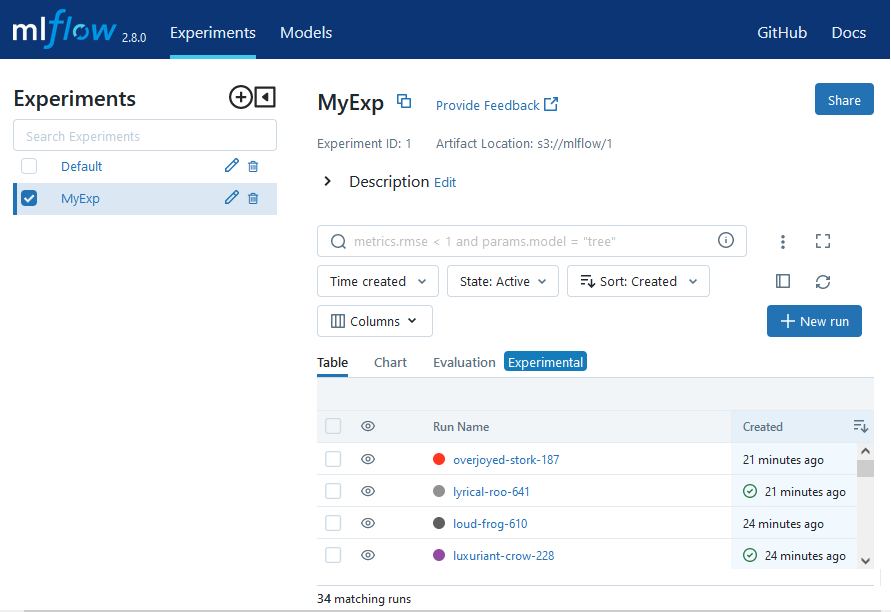


Рис.15 Тесты модели в интерфейсе MLflow

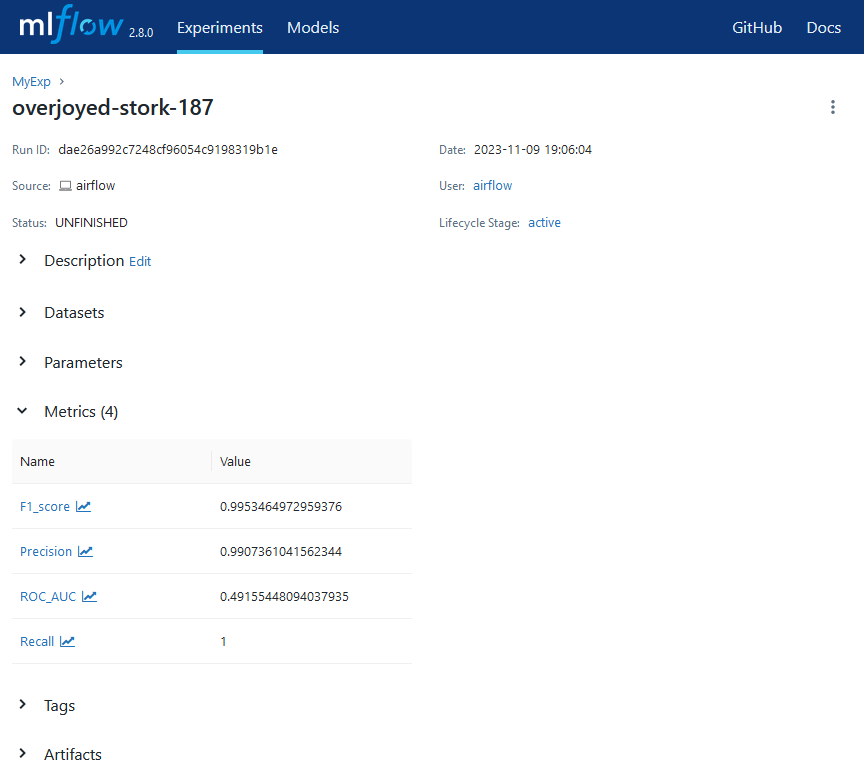


Рис.16 Метрики модели, полученные в результате проведенного теста, в интерфейсе MLflow

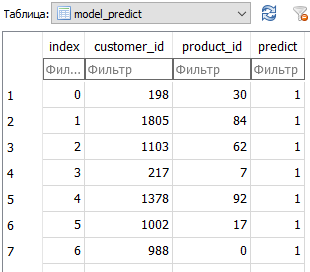


Рис.17 Содержимое таблицы model\_predict - данные, предсказанные ML-моделью

1. **Документация**

Для запуска системы необходимо выполнить следующие шаги:

1. Установить Docker.
2. .В каталоге проекта необходимо выполнить команду “docker compose up -d –build” Проверить, создались контейнеры, можно в программе Docker Desktop.
3. Поместить входные файлы, которые нужно записать в базу данных input\_data.
4. Перейти в браузере по адресу <http://localhost:8080>. Ввести в качестве пользователя и пароля “airflow”
5. На открывшейся странице Airflow в списке DAG щелкнуть по строке “item\_classification”
6. Справа страницы найти кнопку “Trigger” и нажать на нее. Таким образом, запустится выполнение процесса ETL и обучение ML-модели.
7. После успешного завершения всех задач они буду помечены зеленым маркером.В базе mydb.db появятся таблицы, а в папке output\_data будет создан файл ProductsStats.csv.
8. В интерфейсе MLflow по адресу <http://localhost:5001> отобразится обученная ML-модель (вкладка “Models”) и проведенные тесты (вкладка “Experiments”).

Если файлы были помещены в папку archive после обработки, то повторно они уже не обрабатываются (т.е. игнорируются) при последующих запусках ETL-процесса. Если требуется повторно загрузить файлы, то необходимо удалить их из папки archive.

1. **Выводы**

В результате проделанной работы мы получили систему, осуществляющую автоматизированный сбор, обработку и хранение информации, предоставляющую данные для визуализации и содержащую в себе обученную ML-модель, которую можно использовать для прогнозирования и принятия решений. На основе этих данных можно проводить анализ, планирование и корректировку стратегий бизнеса для достижения наилучших показателей.

**Приложение**

**Листинг файла DAG для Airflow Apache**

import json

import pendulum

from airflow.decorators import dag, task

from config import db, root\_path, root\_path\_input\_data, root\_path\_archive, tables\_info,path\_sql\_scripts,output\_data,select\_dwh\_for\_model\_script

import logging

logging.basicConfig(level=logging.DEBUG, filename='myapp.log', format='%(asctime)s %(levelname)s:%(message)s')

logger = logging.getLogger()

logger.setLevel(logging.INFO)

@dag(

schedule=None,

start\_date=pendulum.datetime(2021, 1, 1, tz="UTC"),

catchup=False,

tags=["etl", "classification"],

)

def item\_classification(debug=False):

@task()

def extract\_data\_from\_files\_to\_stage():

from stages.etl.connection import Connection

from stages.etl.extract\_data import Extract\_data

conn = Connection(db).\_create\_connection()

extract = Extract\_data(root\_path, conn, root\_path\_input\_data, root\_path\_archive,tables\_info,db,path\_sql\_scripts)

result=extract.my\_extract\_data\_to\_stg()

'''table = extract.create\_tabel('items')'''

'''return table'''

return f"{result}"

@task()

def transfer\_filesdata\_from\_stage\_to\_dwh(stg\_result):

from stages.etl.connection import Connection

from stages.etl.extract\_data import Extract\_data

conn = Connection(db).\_create\_connection()

extract = Extract\_data(root\_path, conn, root\_path\_input\_data, root\_path\_archive, tables\_info, db,

path\_sql\_scripts)

result = extract.my\_extract\_filesdata\_from\_stg\_to\_dwh()

'''table = extract.create\_tabel('items')'''

'''return table'''

return f"{result}"

@task()

def extract\_web\_data(categories\_and\_webfilters):

from stages.parsing.parcer import Parcer

from stages.etl.connection import Connection

conn = Connection(db).\_create\_connection()

par = Parcer(\*categories\_and\_webfilters)

data = par.take\_website\_data()

table\_name = 'STG\_'+'\_'.join(categories\_and\_webfilters)

logger.info(table\_name)

data.to\_sql(table\_name, con=conn, if\_exists='replace')

return table\_name

@task()

def create\_dwh\_parcer\_data(list\_table):

from stages.sql\_scripts.union\_tables import union\_tabels

from stages.etl.connection import Connection

conn = Connection(db)

logger.info(list\_table)

tables = union\_tabels(list\_table)

s = f"""

Create table if not EXISTS DWH\_ProductsStat as {tables}

"""

logger.info(s)

conn.read\_sql\_script(s, file\_script=False)

return 'DWH\_ProductsStat'

@task()

def create\_outputfile\_for\_dashboards(dwh\_table\_for\_output\_dashboard):

from stages.sql\_scripts.union\_tables import union\_tabels

from stages.etl.connection import Connection

conn = Connection(db)

conn.save\_select\_result\_to\_csv("DWH\_ProductsStat",output\_data,"ProductsStat.csv")

return 'ProductsStat.csv'

@task()

def preprocessing(dwh\_tables):

from stages.sql\_scripts.union\_tables import union\_tabels

from stages.etl.connection import Connection

import pandas as pd

from stages.etl.preprocessing import Preprocessing

print(pd.\_\_version\_\_)

conn = Connection(db).\_create\_connection()

data = pd.read\_sql\_query(select\_dwh\_for\_model\_script, con=conn)

cat\_f = ['gender', 'job\_title','job\_industry\_category','wealth\_segment','state','brand','class','line','size','address','country']

process = Preprocessing(cat\_f)

process\_data = process.process\_data(data)

logger.info(process\_data.columns)

process\_data.to\_sql('model\_data', con=conn, if\_exists='replace')

return 'model\_data'

@task(multiple\_outputs=True)

def train\_test\_split(model\_data):

from stages.etl.connection import Connection

import pandas as pd

from sklearn.model\_selection import train\_test\_split

conn = Connection(db).\_create\_connection()

data = pd.read\_sql\_query(f"""Select \* from {model\_data}""", con=conn)

logger.info(data.columns)

train\_data, test\_data = train\_test\_split(data, random\_state=12345, test\_size=0.2)

logger.info(train\_data.columns)

logger.info(test\_data.columns)

train\_data.to\_sql('train\_data', con=conn, if\_exists='replace')

test\_data.to\_sql('test\_data', con=conn, if\_exists='replace')

return {'test\_data': 'test\_data', 'train\_data': 'train\_data'}

@task(multiple\_outputs=True)

def model\_materialize(model\_name, test\_data):

import mlflow

from stages.etl.connection import Connection

import pandas as pd

import os

d = dict(

AWS\_ACCESS\_KEY\_ID="admin",

AWS\_SECRET\_ACCESS\_KEY="sample\_key",

AWS\_REGION="us - east - 1",

AWS\_BUCKET\_NAME="mlflow",

MYSQL\_DATABASE="mlflow",

MYSQL\_USER="mlflow\_user",

MYSQL\_PASSWORD="mlflow\_password",

MYSQL\_ROOT\_PASSWORD="toor",

MLFLOW\_TRACKING\_URI='http://mlflow:5001',

MLFLOW\_EXPERIMENT\_NAME="MyExp",

MLFLOW\_S3\_ENDPOINT\_URL='http://s3:9000'

)

for i in d:

os.environ[i] = d[i]

logged\_model = f'models:/{model\_name}/None'

conn = Connection(db).\_create\_connection()

# Load model as a PyFuncModel.

loaded\_model = mlflow.pyfunc.load\_model(logged\_model)

test = pd.read\_sql\_query(f"""Select \* from {test\_data}""", con=conn)

predict = loaded\_model.predict(test.drop(columns=['order\_status\_int']))

test['predict'] = predict

test[['customer\_id','product\_id', 'predict']].to\_sql('model\_predict', con=conn, if\_exists='replace')

@task()

def get\_metrics(model\_name, test\_data):

import mlflow

from stages.etl.connection import Connection

from stages.models.linear\_model import LinearModel

import pandas as pd

import os

d = dict(

AWS\_ACCESS\_KEY\_ID="admin",

AWS\_SECRET\_ACCESS\_KEY="sample\_key",

AWS\_REGION="us - east - 1",

AWS\_BUCKET\_NAME="mlflow",

MYSQL\_DATABASE="mlflow",

MYSQL\_USER="mlflow\_user",

MYSQL\_PASSWORD="mlflow\_password",

MYSQL\_ROOT\_PASSWORD="toor",

MLFLOW\_TRACKING\_URI='http://mlflow:5001',

MLFLOW\_EXPERIMENT\_NAME="MyExp",

MLFLOW\_S3\_ENDPOINT\_URL='http://s3:9000'

)

for i in d:

os.environ[i] = d[i]

logged\_model = f'models:/{model\_name}/None'

conn = Connection(db).\_create\_connection()

# Load model as a PyFuncModel.

loaded\_model = mlflow.sklearn.load\_model(logged\_model)

mmodel = LinearModel()

mmodel.model = loaded\_model

test = pd.read\_sql\_query(f"""Select \* from {test\_data}""", con=conn)

metrics = mmodel.get\_metrics(test.drop(columns=['order\_status\_int','index','level\_0']), test['order\_status\_int'])

for j in metrics:

mlflow.log\_metric(j, metrics[j])

@task()

def model\_fit(train\_data, test\_data):

import os

from stages.etl.connection import Connection

import pandas as pd

from stages.models.linear\_model import LinearModel

import mlflow

from mlflow.models import infer\_signature

d = dict(

AWS\_ACCESS\_KEY\_ID="admin",

AWS\_SECRET\_ACCESS\_KEY="sample\_key",

AWS\_REGION="us - east - 1",

AWS\_BUCKET\_NAME="mlflow",

MYSQL\_DATABASE="mlflow",

MYSQL\_USER="mlflow\_user",

MYSQL\_PASSWORD="mlflow\_password",

MYSQL\_ROOT\_PASSWORD="toor",

MLFLOW\_TRACKING\_URI='http://mlflow:5001',

MLFLOW\_EXPERIMENT\_NAME="MyExp",

MLFLOW\_S3\_ENDPOINT\_URL='http://s3:9000'

)

for i in d:

os.environ[i] = d[i]

conn = Connection(db).\_create\_connection()

train = pd.read\_sql\_query(f"""Select \* from {train\_data}""", con=conn)

logger.info('train\_data')

logger.info(train.columns)

model = LinearModel()

with mlflow.start\_run() as run:

model.fit(train.drop(columns=['order\_status\_int','level\_0', 'index']), train['order\_status\_int'])

test = pd.read\_sql\_query(f"""Select \* from {test\_data}""", con=conn)

logger.info('test\_data')

logger.info(train.columns)

preds = model.model.predict(test.drop(columns=['order\_status\_int','level\_0', 'index']))

signature = infer\_signature(train.drop(columns=['order\_status\_int','level\_0', 'index']), preds)

mlflow.sklearn.log\_model(model.model, "model", signature=signature, registered\_model\_name='LinearModel')

return 'LinearModel'

stg\_file\_result = extract\_data\_from\_files\_to\_stage()

dwh\_file\_result = transfer\_filesdata\_from\_stage\_to\_dwh(stg\_file\_result)

item\_category = ['shlemy', 'flyagi', 'ryukzaki\_i\_sumki']

item\_filter = ['new', 'discount', 'popular']

list3 = [[i, str(j)] for i in item\_category for j in item\_filter]

website\_tables = extract\_web\_data.expand(categories\_and\_webfilters=list3)

dwh\_table\_website = create\_dwh\_parcer\_data(website\_tables)

output\_dashboard\_file = create\_outputfile\_for\_dashboards(dwh\_table\_website)

prep = preprocessing(dwh\_file\_result)

train\_test = train\_test\_split(prep)

model = model\_fit(train\_test['train\_data'], train\_test['test\_data'])

model\_mat = model\_materialize(model, train\_test['test\_data'])

get\_metrics(model, train\_test['test\_data'])

prod = item\_classification(debug=False)