

Департамент образования и науки города Москвы
Государственное автономное образовательное учреждение высшего
образования города Москвы
«Московский городской педагогический университет»
Институт цифрового образования
Департамент информатики, управления и технологий

Выполнила: st_95

Вебинар от 21.03.2025

Бизнес кейс «Umbrella»

Проектный практикум по разработке ETL-решений

Направление подготовки

38.03.05 Бизнес-информатика

Профиль подготовки

Аналитика данных и эффективное управление

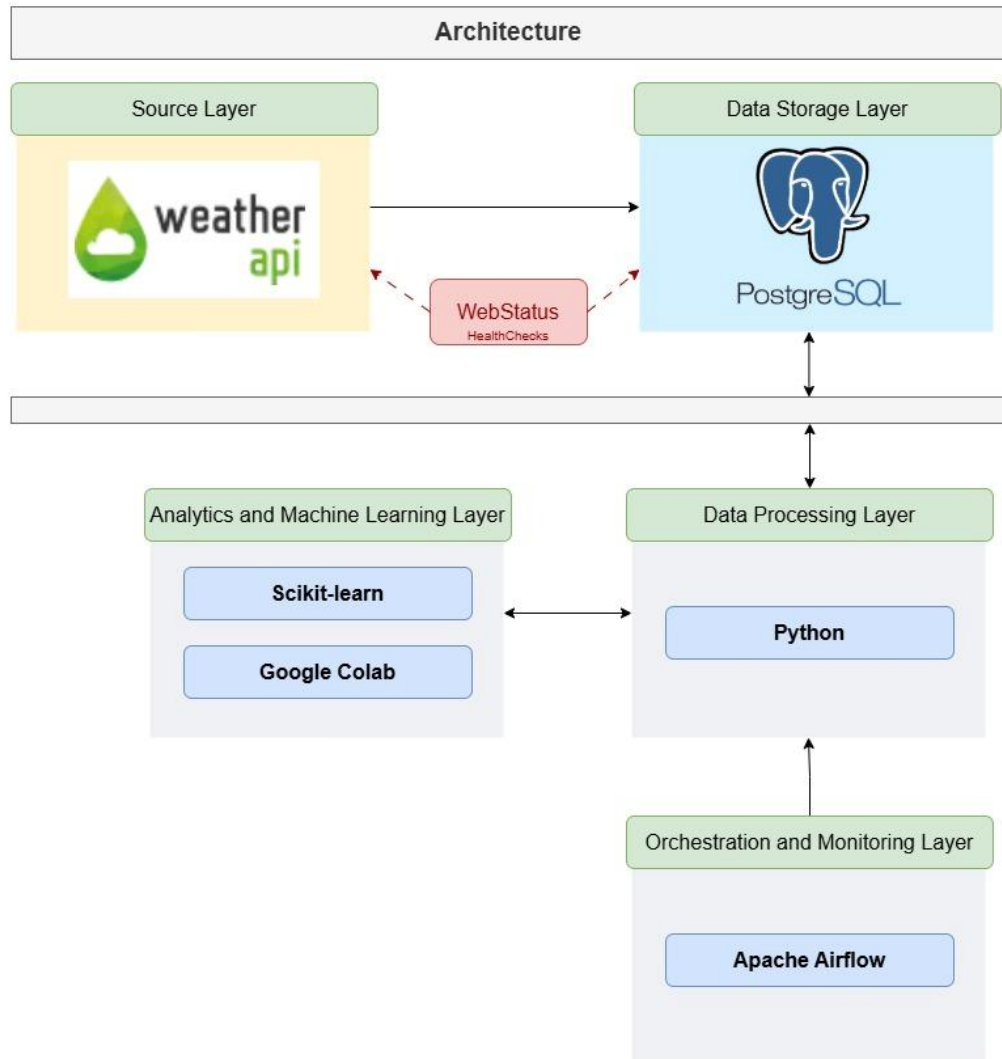
Москва

2025

Вариант 8

Задание 1	Задание 2	Задание 3
Получить прогноз в Токио на 3 дня	Преобразовать JSON-ответ в таблицу pandas	Сохранить в формате Excel

Архитектура:



Ход работы:

1. Клонирование репозитория (Рисунок 1).

```
mgpu@mgpu-VirtualBox:~$ git clone https://github.com/BosenkoTM/workshop-on-ETL.git
Cloning into 'workshop-on-ETL'...
remote: Enumerating objects: 563, done.
remote: Counting objects: 100% (453/453), done.
remote: Compressing objects: 100% (394/394), done.
remote: Total 563 (delta 222), reused 59 (delta 32), pack-reused 110 (from 1)
Receiving objects: 100% (563/563), 5.82 MiB | 6.67 MiB/s, done.
Resolving deltas: 100% (260/260), done.
mgpu@mgpu-VirtualBox:~$ ls
Desktop    Downloads  Pictures   snap       thinclient_drives  workshop-on-ETL
Documents  Music      Public     Templates  Videos
```

Рисунок 1

2. Получение api key на Free Weather Api (Рисунок 2).

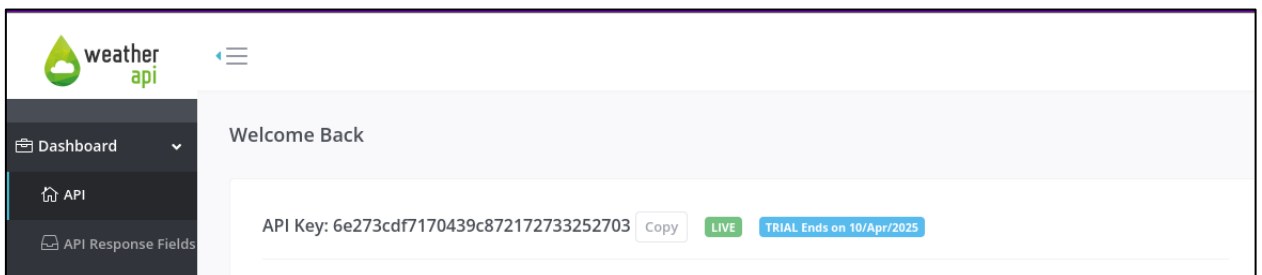


Рисунок 2

3. В файле DAG «real_umbrella.py» указываем значение полученного API ключа (Рисунок 3).

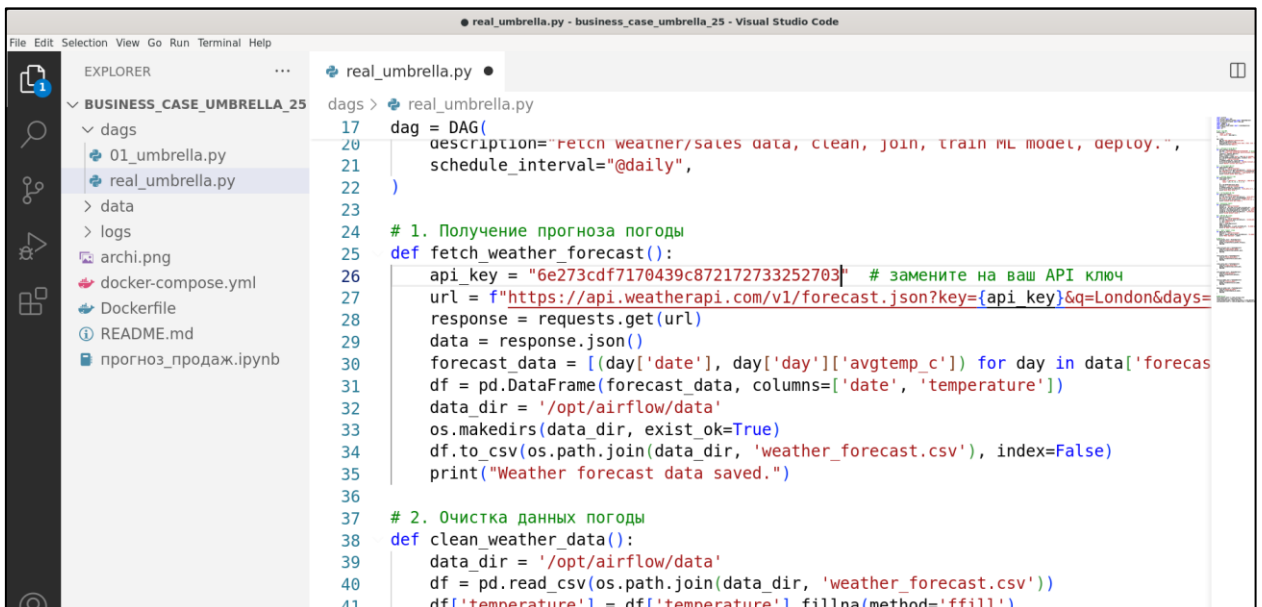


Рисунок 3

4. Далее меняем локацию «q» на Токио и значение «days» на 3, в соответствии с 8 вариантом (Рисунок 4).

```
12 default_args = {
13     'owner': 'airflow',
14     'start_date': days_ago(1),
15 }
16
17 dag = DAG(
18     dag_id="real_umbrella_containerized",
19     default_args=default_args,
20     description="Fetch weather/sales data, clean, join, train ML model, deploy.",
21     schedule_interval="@daily",
22 )
23
24 # 1. Получение прогноза погоды
25 def fetch_weather_forecast():
26     api_key = "6e273cdf7170439c872172733252703" # замените на ваш API ключ
27     url = f"https://api.weatherapi.com/v1/forecast.json?key={api_key}&q=Tokyo&days=3"
28     response = requests.get(url)
29     data = response.json()
30     forecast_data = [(day['date'], day['day']['avgtemp_c']) for day in data['forecast']['forecastday']]
31     df = pd.DataFrame(forecast_data, columns=['date', 'temperature'])
32     data_dir = '/opt/airflow/data'
```

Рисунок 4

5. По варианту необходимо преобразовать ответ json в таблицу pandas (Рисунок 5).

```
# 1. Получение прогноза погоды
def fetch_weather_forecast():
    api_key = "6e273cdf7170439c872172733252703" # замените на ваш API ключ
    url = f"https://api.weatherapi.com/v1/forecast.json?key={api_key}&q=Tokyo&days=3"
    response = requests.get(url)
    data = response.json()
    forecast_data = [(day['date'], day['day']['avgtemp_c']) for day in data['forecast']['forecastday']]
    df = pd.DataFrame(forecast_data, columns=['date', 'temperature'])
    data_dir = '/opt/airflow/data'
    os.makedirs(data_dir, exist_ok=True)
    df.to_excel(os.path.join(data_dir, 'weather_forecast.xlsx'), index=False, engine='openpyxl')
    print("Weather forecast data saved.")
```

Рисунок 5

6. По варианту необходимо сохранять данные в формате Excel, поэтому была изменена строка с сохранением данных (Рисунок 6).

```
# 1. Получение прогноза погоды
def fetch_weather_forecast():
    api_key = "6e273cdf7170439c872172733252703" # замените на ваш API ключ
    url = f"https://api.weatherapi.com/v1/forecast.json?key={api_key}&q=Tokyo&days=3"
    response = requests.get(url)
    data = response.json()
    forecast_data = [(day['date'], day['day']['avgtemp_c']) for day in data['forecast']['forecastday']]
    df = pd.DataFrame(forecast_data, columns=['date', 'temperature'])
    data_dir = '/opt/airflow/data'
    os.makedirs(data_dir, exist_ok=True)
    df.to_excel(os.path.join(data_dir, 'weather_forecast.xlsx'), index=False, engine='openpyxl')
    print("Weather forecast data saved.")
```

Рисунок 6

Также обновим строку с чтением данных о прогнозе погоды на формат данных excel (Рисунок 7).

```
# 2. Очистка данных погоды
def clean_weather_data():
    data_dir = '/opt/airflow/data'
    df = pd.read_excel(os.path.join(data_dir, 'weather_forecast.xlsx'), engine='openpyxl')
    df['temperature'] = df['temperature'].fillna(method='ffill')
    df.to_csv(os.path.join(data_dir, 'clean_weather.csv'), index=False)
    print("Cleaned weather data saved.")
```

Рисунок 7

7. Добавим импорт openpyxl в DockerFile (Рисунок 8).

```
Dockerfile
1 FROM apache/airflow:slim-2.8.1-python3.11
2
3 USER airflow
4
5 # Устанавливаем необходимые Python-библиотеки
6 RUN pip install --no-cache-dir \
7     pandas \
8     scikit-learn \
9     joblib \
10    requests \
11    azure-storage-blob==12.8.1 \
12    psycpg2-binary \
13    "connexion[swagger-ui]" \
14    openpyxl
```

Рисунок 8

8. Настройка данных о продажах:

- Проверим погоду в Токио на ближайшие три дня (Рисунок 9).

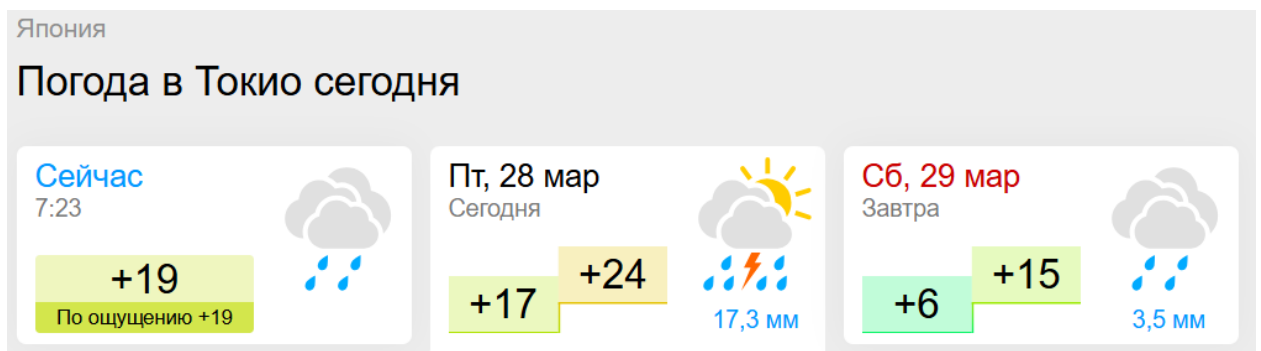


Рисунок 9

Видим, что следующие дни будут дождливыми, поэтому можно сделать предположение, что продажи зонтов вырастут.

- Изменим даты в коде на 28.03.2025, 29.03.2025 и 30.03.2025, а также увеличим количество продаж исходя из того, что в Токио будут дожди (Рисунок 10).

```
# 3. Получение данных продаж
def fetch_sales_data():
    sales_data = {
        'date': ['2025-03-28', '2025-03-29', '2025-03-30'],
        'sales': [10, 20, 25]
    }
    df = pd.DataFrame(sales_data)
    data_dir = '/opt/airflow/data'
    os.makedirs(data_dir, exist_ok=True)
    df.to_csv(os.path.join(data_dir, 'sales_data.csv'), index=False)
    print("Sales data saved.")
```

Рисунок 10

9. Проверка работающих контейнеров (Рисунок 11).

```
mgpu@mgpu-VirtualBox:~/workshop-on-ETL/business_case_umbrella_25$ sudo docker ps
[sudo] password for mgpu:
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS          NAMES
mgpu@mgpu-VirtualBox:~/workshop-on-ETL/business_case_umbrella_25$
```

Рисунок 11

10. Собираем контейнер из текущей директории и присваиваем ему тег (Рисунок 12).

```
mgpu@mgpu-VirtualBox:~/workshop-on-ETL/business_case_umbrella_25$ sudo docker build -t custom-airflow:slim-2.8.1-python3.11 .
[+] Building 117.9s (7/7) FINISHED
=> [internal] load build definition from Dockerfile                                docker:default    0.0s
=> => transferring dockerfile: 568B                                              0.0s
=> [internal] load metadata for docker.io/apache/airflow:slim-2.8.1-python3.11    1.3s
=> [internal] load .dockerignore                                                  0.0s
=> transferring context: 2B                                                       0.0s
=> [1/3] FROM docker.io/apache/airflow:slim-2.8.1-python3.11@sha256:751b4d59e82e44a023e202fc1553106c25f2e2682c07db1a6 15.5s
```

Рисунок 12

Собираем и запускает все сервисы, описанные в docker-compose.yml (Рисунок 13).

```
mgpu@mgpu-VirtualBox:~/workshop-on-ETL/business_case_umbrella_25$ sudo docker compose up --build
[+] Running 7/7
✓ Network business_case_umbrella_25_default      Created
✓ Volume "business_case_umbrella_25_postgres_data" Created
✓ Volume "business_case_umbrella_25_logs"         Created
✓ Container business_case_umbrella_25-postgres-1 Created
✓ Container business_case_umbrella_25-init-1      Created
✓ Container business_case_umbrella_25-scheduler-1 Created
✓ Container business_case_umbrella_25-webserver-1 Created
Attaching to init-1, postgres-1, scheduler-1, webserver-1
```

Рисунок 13

11. Проверка доступности к apache airflow по адресу <http://localhost:8080/> (Рисунок 14).

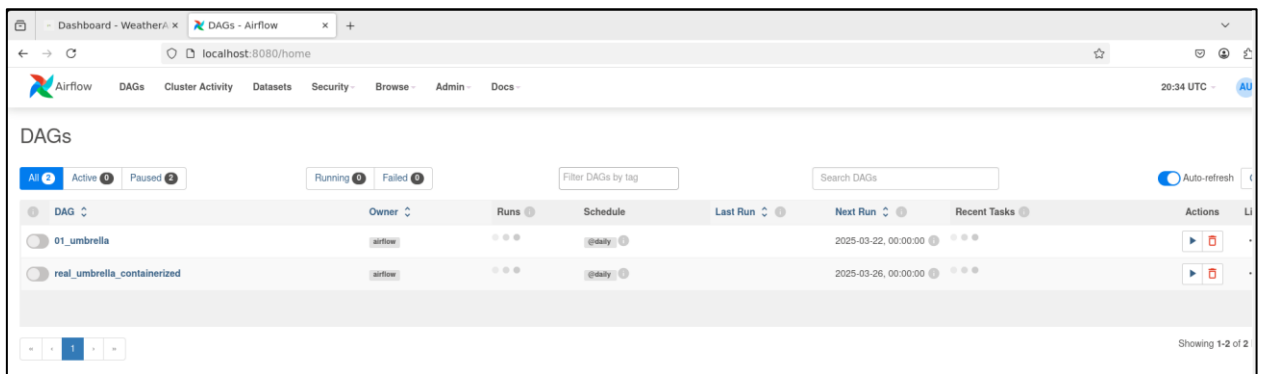


Рисунок 14

12. Запускаем модельный даг, иллюстрирующий вариант использования бизнес-кейса «Umbrella» в модельном виде (Рисунок 15).

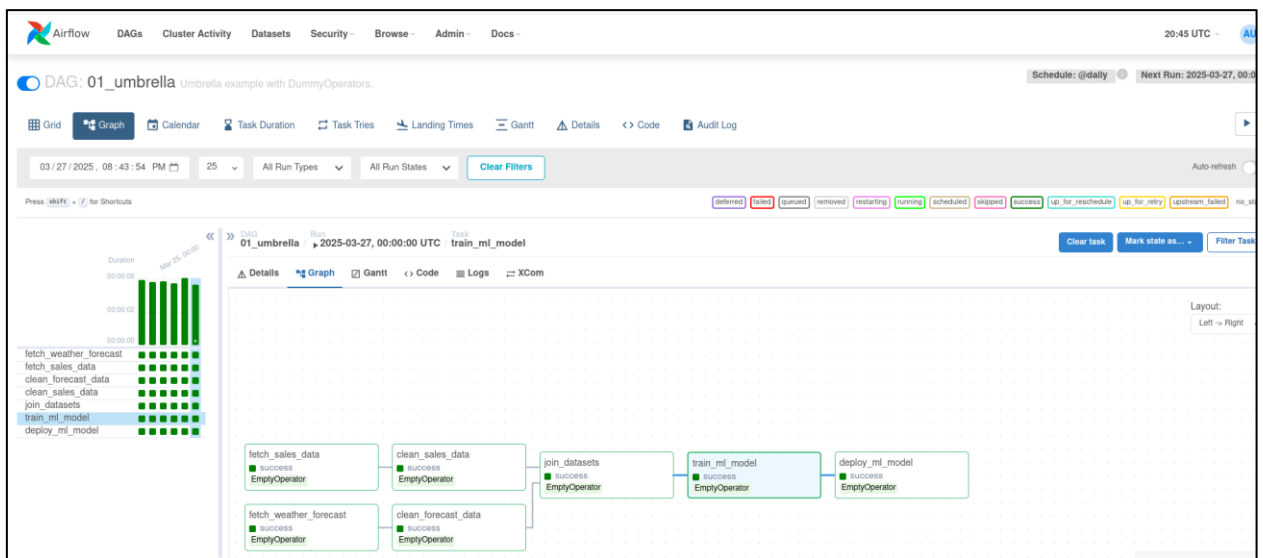


Рисунок 15

13. Запуск дага `real_umbrella.py`, иллюстрирующего вариант использования бизнес-кейса «Umbrella», который автоматизирует весь pipeline обработки погодных и продажных данных, обучения модели машинного обучения и её «развёртывания» (Рисунок 16).

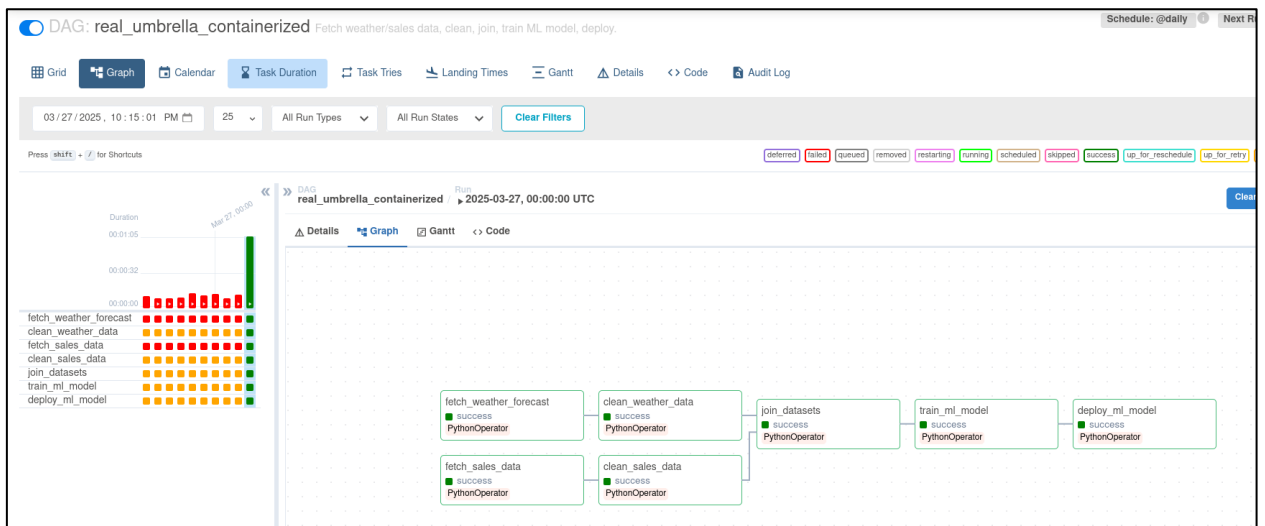


Рисунок 16

Данные о прогнозе погоды были успешно получены и обработаны (Рисунок 17).

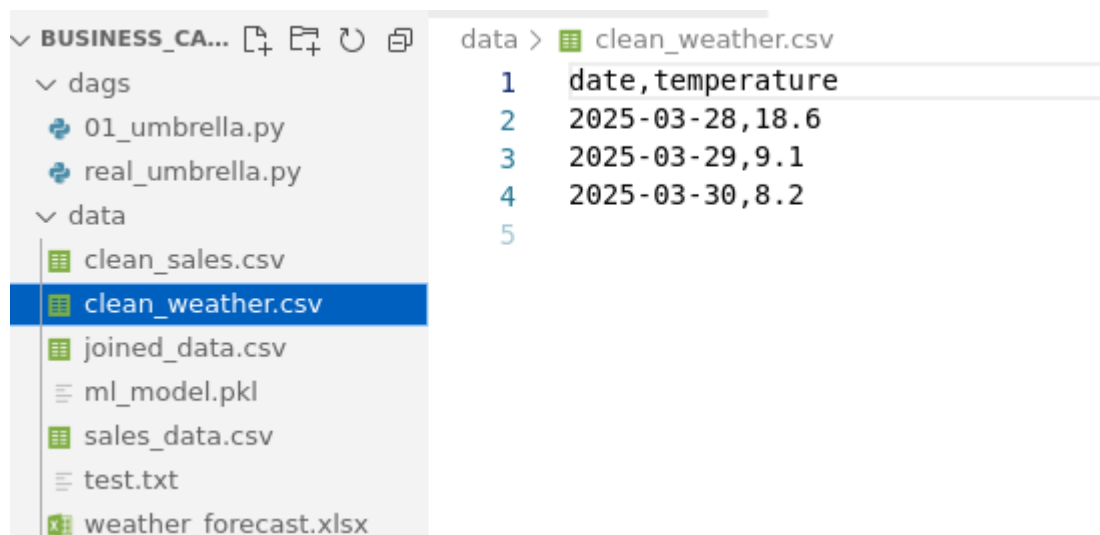


Рисунок 17

А также успешно объединены с данными о продажах (Рисунок 18).

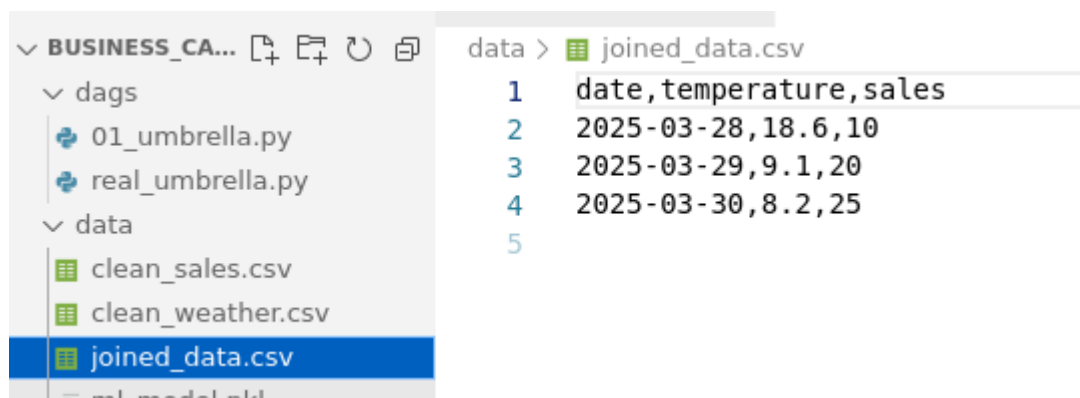
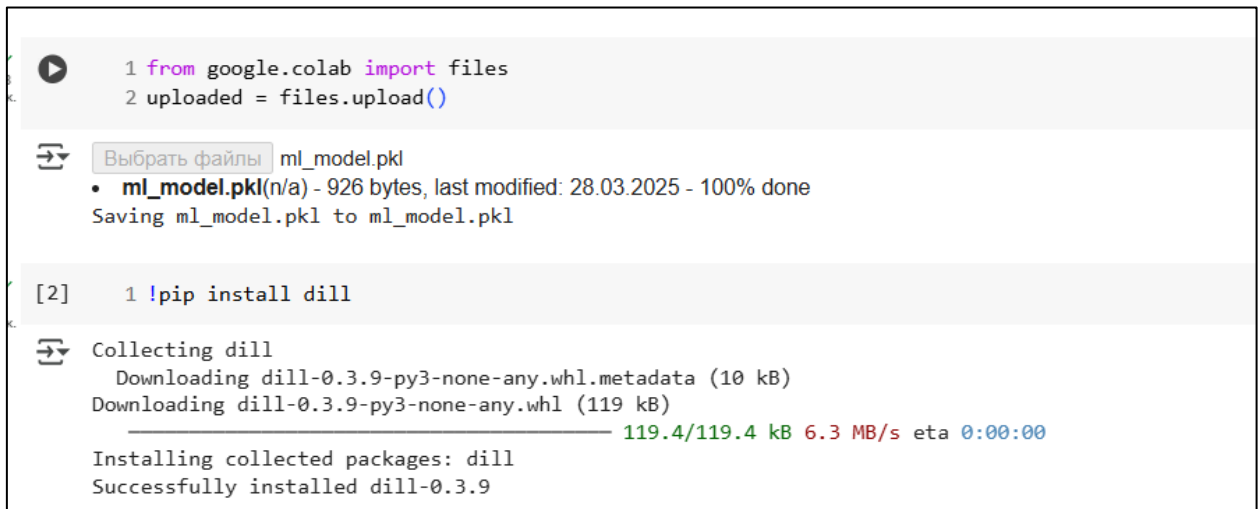


Рисунок 18

Используя Google Colab и исходный файл кода, на основе обученной модели спрогнозируем количество продаж зонтов в зависимости от температуры:



The screenshot shows a Google Colab notebook with two code cells. The first cell contains code to upload a file from Google Drive. The second cell contains code to install the 'dill' package. The output of the second cell shows the download progress and successful installation of dill-0.3.9.

```
1 from google.colab import files
2 uploaded = files.upload()

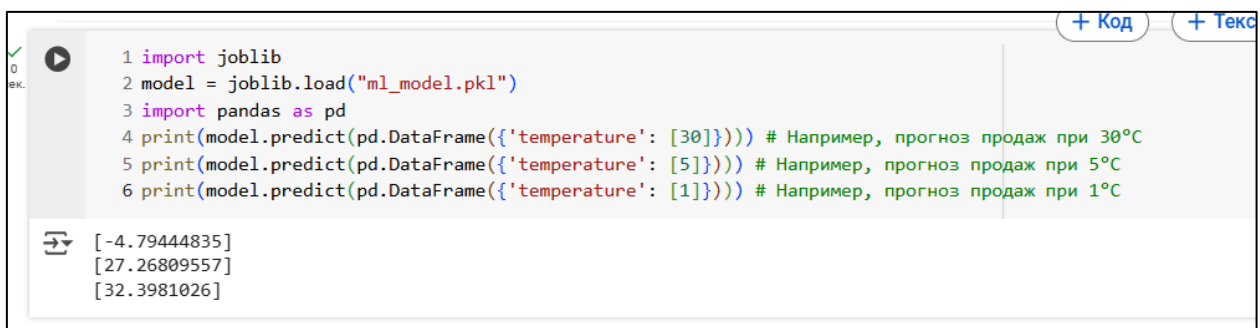
Выбрать файлы ml_model.pkl
• ml_model.pkl(n/a) - 926 bytes, last modified: 28.03.2025 - 100% done
Saving ml_model.pkl to ml_model.pkl

[2] 1 !pip install dill

Collecting dill
  Downloading dill-0.3.9-py3-none-any.whl.metadata (10 kB)
  Downloading dill-0.3.9-py3-none-any.whl (119 kB)
    119.4/119.4 kB 6.3 MB/s eta 0:00:00
Installing collected packages: dill
Successfully installed dill-0.3.9
```

Рисунок 19

Проверим результат модели при уменьшении температуры, результат показал, что уменьшении температуры количество продаж будет возрастать (Рисунок 20).



The screenshot shows a Google Colab notebook with a single code cell. The code imports joblib, loads the model, imports pandas, and prints the predicted sales for three different temperatures: 30°C, 5°C, and 1°C. The output shows that as the temperature decreases, the predicted sales increase.

```
1 import joblib
2 model = joblib.load("ml_model.pkl")
3 import pandas as pd
4 print(model.predict(pd.DataFrame({'temperature': [30]}))) # Например, прогноз продаж при 30°C
5 print(model.predict(pd.DataFrame({'temperature': [5]}))) # Например, прогноз продаж при 5°C
6 print(model.predict(pd.DataFrame({'temperature': [1]}))) # Например, прогноз продаж при 1°C

[-4.79444835]
[27.26809557]
[32.3981026]
```

Рисунок 20

Такой результат получился из-за того, что модель обучалась всего на трёх записях (!), и если посмотреть на данные, на которых обучалась модель (см. Рисунок 18) можно увидеть, что при уменьшении температуры растут продажи, отсюда такой результат.

Для более точного обучения модели необходимо больше данных, а также иметь реальные данные о продажах, а не взятые из головы.

Описание основной архитектуры Airflow:

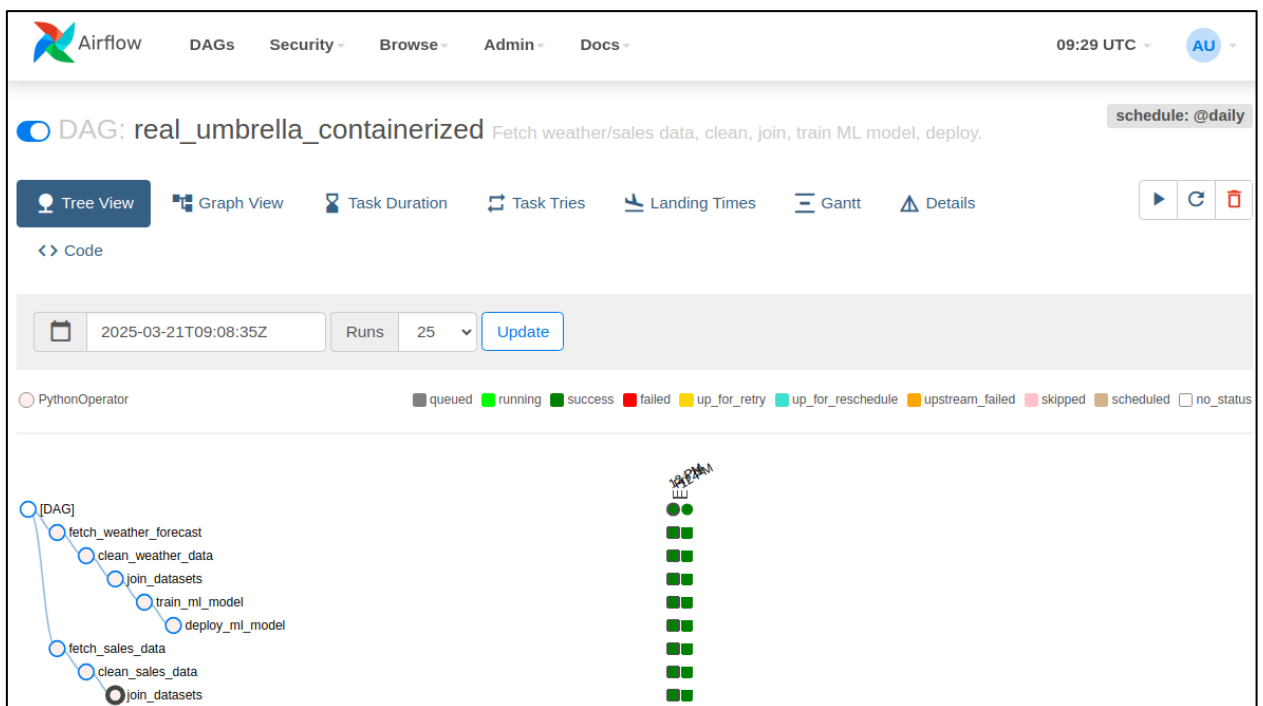
Главная страница при входе представляет из себя список всех доступных дагов:

DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Action
<input type="checkbox"/> 01_umbrella	airflow	4	@daily	2025-03-21, 09:07:49	7	▶ ↺
<input checked="" type="checkbox"/> real_umbrella_containerized	airflow	2	@daily	2025-03-21, 09:08:35	7	▶ ↺

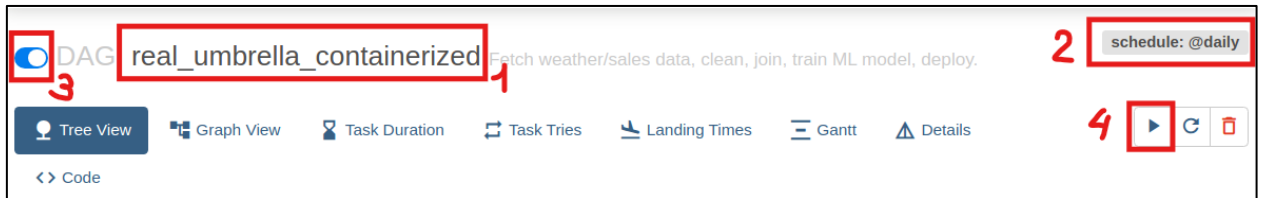
Showing 1-2 of 2 DAGs

По каждому дагу отображается основная информация по нему: запущен ли, наименование, владелец, количество выполненных раз (при том, зеленым отображаются успешно выполненные даги, красным – в случае ошибки), также частота выполнения, время последнего запуска и количество задач в даге.

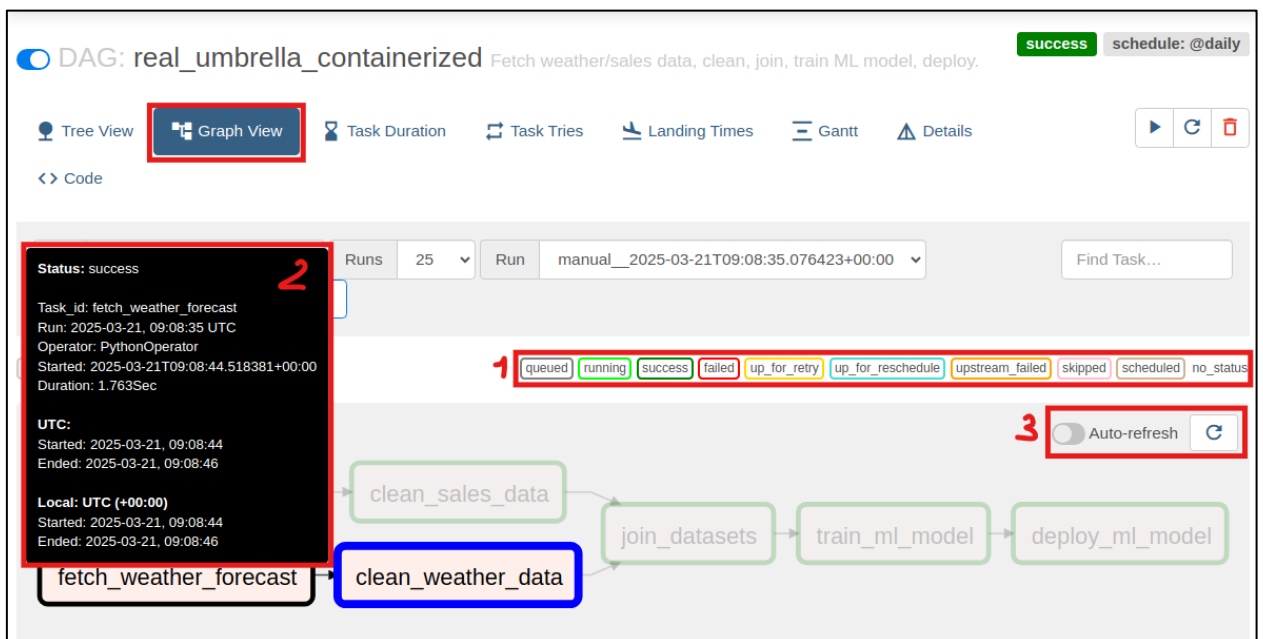
Информационная карта дага выглядит следующим образом:



Соответственно, на верхней панели отображается наименование графа (1) и интервал его запуска (2). Также, доступен свитч для включения/выключения дага (3) и кнопка для триггера дага (4):



Далее ниже доступна вариация представления графа: в виде дерева задач, в виде графа, просмотр информации о длительности выполнения и другое. Наиболее удобным вариантом для меня является отображение в виде графа:



В данном случае отображаются задачи в виде графа. Каждая задача подсвечивается цветом, соответствующим текущему статусу выполнения (1). Помимо этого, при наведении на task в открывшемся модальном окошке (2) можно изучить всю информацию о нем: id, оператор, время запуска и длительность выполнения.

Для отображения актуальных статусов выполнения задач можно настроить автообновление, либо же запустить принудительное обновление (3).