Assignment 3

Group:
Max van Klinski
Simon Stephan

Project structure

We wrote some test functions to test our Python methods. That's why we included a `test` directory in our solution. Here is our solution explained in detail:

- **dags**: Directory where Airflow DAGs (Directed Acyclic Graphs) are stored.
  - `OnlineRetailAnalysize.py`: Python script defining the DAG for downloading, cleaning, transforming, and saving the online retail dataset.
  - `utils.py`: Contains all the functions used for the tasks. Further details are explained later.
- **Snapshots and Logs**: Contains all logs of our tasks copied from Airflow and a snapshot of the collection from our MongoDb.
- **tests**: Contains everything to test our used functions locally.
  - **data**: Directory containing data files. These files are generated by our ETL.
    - `Online_Retail_features.csv`: Original dataset before cleaning.
    - `Online_Retail_features_cleaned.csv`: Cleaned dataset after applying data cleaning operations.
    - `Online_Retail_features_transformed.csv`: Dataset after applying data cleaning and transformation.
  - `requirements.txt`: Contains all necessary packages to run the tests.
  - `test_clean_data.py`
  - `test_download_data.py`
  - `test_transform_data.py`
  - `OnlineRetailAnalysize.py`: Copy from the `dags` directory.
  - `utils.py`: Copy from the `dags` directory.

ETL

In this section, we explain how our ETL pipeline works:

1. Install Needed Packages

```
pip_install_task = BashOperator(
    task_id='install_libraries',
    bash_command="pip install ucimlrepo pymongo",
    dag=dag,
)
```

This task installs the required packages (`ucimlrepo` and `pymongo`) for our subsequent tasks using `pip`.

2. Download the Data

```
download_task = PythonOperator(
    task_id='download_online_retail_task',
    python_callable=download_online_retail,
    dag=dag,
)
```

This task downloads the online retail dataset using the `utils.download_online_retail` function from the UCI Machine Learning Repository and stores it under `data/Online_Retail_features.csv`.

3. Clean the Data

```
clean_data_task = PythonOperator(
    task_id='clean_data_task',
    python_callable=clean_data,
    dag=dag,
)
```

This task loads the downloaded dataset and cleans it using `utils.clean_data`. The cleaning process includes dropping NaN values, removing duplicates, and converting the `InvoiceDate` column to datetime format. The cleaned data is saved as `data/Online_Retail_features_cleaned.csv`.

4. Transform the Data

```
transform_data_task = PythonOperator(
    task_id='transform_data_task',
    python_callable=add_total_price,
    dag=dag,
)
```

This task loads the cleaned dataset and applies a transformation by adding a new `TotalPrice` column using `utils.add_total_price`. The `TotalPrice` is computed as the product of `Quantity` and `UnitPrice`, rounded to two decimal places to represent currency. The transformed data is saved as `Online_Retail_features_transformed.csv`.

5. Save the Data to MongoDB

```
load_to_mongodb_task = PythonOperator(
    task_id='load_to_mongodb_task',
    python_callable=load_to_mongodb,
    dag=dag,
)
```

This task loads the transformed dataset and stores its contents into a MongoDB instance using `utils.load_to_mongodb`. It connects to MongoDB using `pymongo`, specifying the host, port, database, username, and password. We added the MongoDB service to our Docker Compose configuration:

```
docker
  mongo:
    image: mongo:latest
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: mongo_admin
      MONGO_INITDB_ROOT_PASSWORD: password
      MONGO_INITDB_DATABASE: assignment3
    volumes:
      - mongodb-data:/data/db
    healthcheck:
      test: ["CMD", "mongo", "--eval", "db.adminCommand('ping')"]
      interval: 10s
      timeout: 5s
      retries: 3
    restart: always
```

After loading the data and connecting to the database, we use the `insert_many` function to insert the data into our MongoDB database.

The data from `Online_Retail_features_transformed.csv` is inserted into the `online_retail` collection in the `assignment3` database.

6. Send summary email

```
send_summary_email_task = PythonOperator(
    task_id='send_summary_email_task',
    python_callable=send_summary_email,
    provide_context=True,
    dag=dag,
)
```

This task sends an mail that the ETL is finsished and the data can be used for Machine Learning.

7. Task with error

```
task_with_error = PythonOperator(
    task_id='task_with_error',
    python_callable=throw_error,
    dag=dag,
)
```

To show that the error mail works we added a task at the end that throws an exception. You should get a notification email that the task did not work containing a link to the log of this task.

Triggering the ETL Pipeline

For point 6 "Triggering the ETL Pipeline" from the assignment sheet, we schedule the DAG to run daily:

```
dag = DAG(
    'download_online_retail',
    default_args=default_args,
    description='Download Online Retail dataset from UCI Machine Learning Repository using u
    schedule_interval='@daily',
)
```

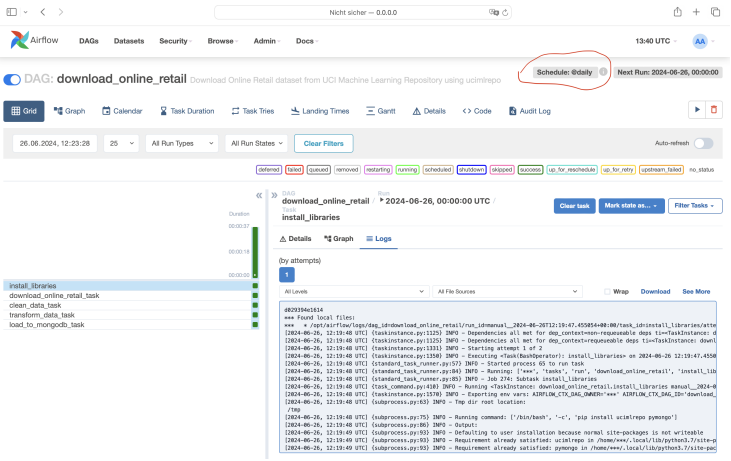

Figure 1: Daily Schedule

Monitoring

For the mail notification we used the build in function of airflow. We set the SMTP variables in the docker compose:

```
AIRFLOW__SMTP__SMTP_HOST: smtp.gmail.com
AIRFLOW__SMTP__SMTP_STARTTLS: 'True'
AIRFLOW__SMTP__SMTP_SSL: 'False'
AIRFLOW__SMTP__SMTP_USER: stephansimon324@gmail.com
AIRFLOW__SMTP__SMTP_PASSWORD: <password>
AIRFLOW__SMTP__SMTP_PORT: '587'
AIRFLOW__SMTP__SMTP_MAIL_FROM: stephansimon324@gmail.com
```

The Readme describes how to redirect the notifications to your email.

Email notifications

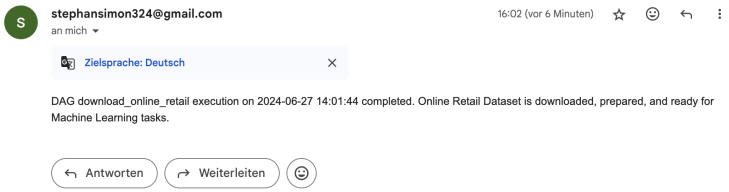

Figure 2: Email Summary

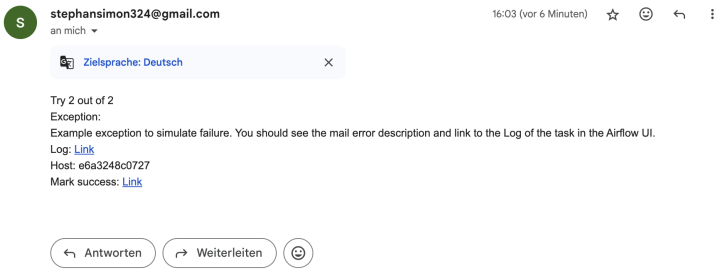

Figure 3: Email Error