Homework 4: simple Airflow pipeline

Installation and initial configuration for Docker is explained in HW3
https://github.com/chap4ev/hse_dc_2024/tree/main/hw3

Let's build a simple data processing pipeline:

1. download dataset from huggingface and save it locally as csv.
   I choose titanic dataset as a classic ML dataset
2. read csv and calculate some statistics
3. write data to local file

Configuring Docker

We will need some additional modules:
datasets - to download dataset from huggingface
pandas - to proces this dataset

To add these modules we need to build our own docker image.

Add ./Dockerfile:

```
FROM apache/airflow:2.3.0
COPY requirements.txt /
RUN pip install --no-cache-dir "apache-airflow==${AIRFLOW_VERSION}" -r
/requirements.txt
```

And ./requirements.txt :

```
datasets
pandas
```

Now change docker-compose.yaml so it will not use predefined image and build its own:

```
 # image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.3.0}
 build: .
```

Lets build our image
`docker-compose build`

Additionally we will need a separate volume to store our data:
adding data volume to docker-compose.yaml

```
 volumes:
   - ./dags:/opt/airflow/dags
   - ./logs:/opt/airflow/logs
   - ./plugins:/opt/airflow/plugins
   - ./data:/data
```

Initialize airflow:
`docker-compose up airflow-init`

And run airflow services:
`docker-compose up`

Writing DAG

DAG definition

The code is placed inside ./dags/ folder as a separate .py file

```python
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.utils.dates import days_ago

from datetime import datetime, timedelta
import os

from datasets import load_dataset
import pandas as pd


default_args = {
    'owner': 'airflow',
    'start_date': days_ago(0),
    'depends_on_past': False,
}


def download_titanic_dataset(path_to_file):
    dataset = load_dataset('lewtun/titanic', split='train')
    df = pd.DataFrame(dataset)
    df.to_csv(path_to_file, index=False)


def calculate_average_age(path_to_file):
    df = pd.read_csv(path_to_file)
    average_age = df['Age'].mean()
    return average_age


def calculate_survival_chance(path_to_file):
    df = pd.read_csv(path_to_file)
    survival_rate = df['Survived'].mean()
    return survival_rate


def save_results(average_age, survival_chance, path_to_file):
    with open(path_to_file, 'w') as f:
        f.write(
            f'Average Age: {average_age}\n'
            f'Survival chance: {survival_chance}'
        )


with DAG(
    'titanic_data_analysis_with_datasets',
    description='Download Titanic dataset, calculate average age and survival
chance, and save results',
    schedule_interval="@once",
    catchup=False,
    default_args=default_args,
) as dag:

    download_dataset = PythonOperator(
        task_id='download_titanic_dataset',
        python_callable=download_titanic_dataset,
        op_args=['/data/titanic.csv'],
    )

    calculate_age = PythonOperator(
        task_id='calculate_average_age',
        python_callable=calculate_average_age,
        op_args=['/data/titanic.csv'],
    )

    calculate_survival = PythonOperator(
        task_id='calculate_survival_chance',
        python_callable=calculate_survival_chance,
        op_args=['/data/titanic.csv'],
    )

    save_age_results = PythonOperator(
        task_id='save_results',
        python_callable=save_results,
        op_args=[calculate_age.output, calculate_survival.output,
'/data/results.txt'],
    )

    download_dataset >> [calculate_age, calculate_survival] >> save_age_results
```

This DAG performs a sequence of 4 PythonOperator tasks:

1. `download_titanic_dataset`:

   This task downloads titanic dataset from huggingface and saves it to specified local file

2. `calculate_age`:

   This task reads a local file with a specified dataset and calculates average passengers age.

3. `calculate_survival`:

   This task reads a local file with a specified dataset and calculates the chance of survival for passengers.

4. `save_age_results`:

   This task takes as input the results of calculations of the previous two tasks and writes them to a local file on disk.

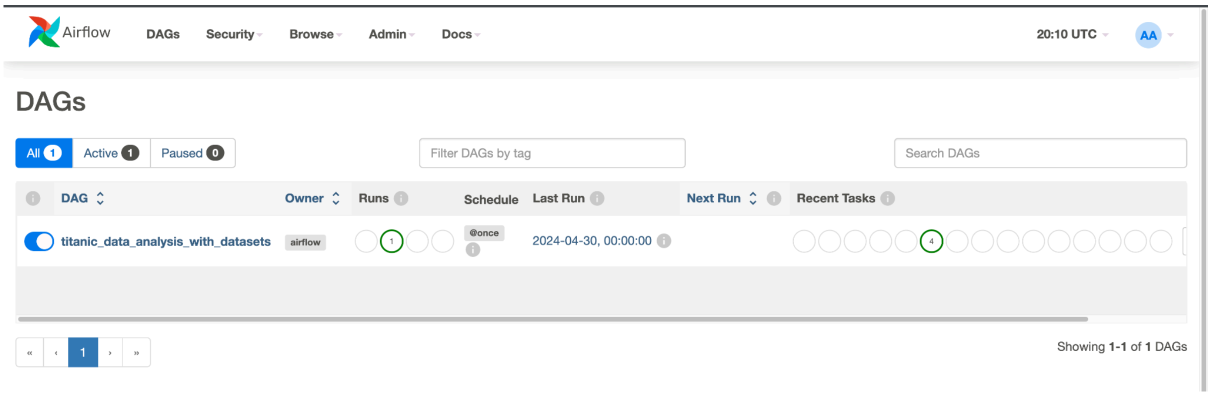
The two counting tasks are independent of each other, so they run in parallel

```
download_dataset >> [calculate_age, calculate_survival] >> save_age_results
```

The DAG triggers by hand and does not contain any schedule

Execute DAG

We can see our DAG ready to be executed.



DAGs

All 1 | Active 1 | Paused 0

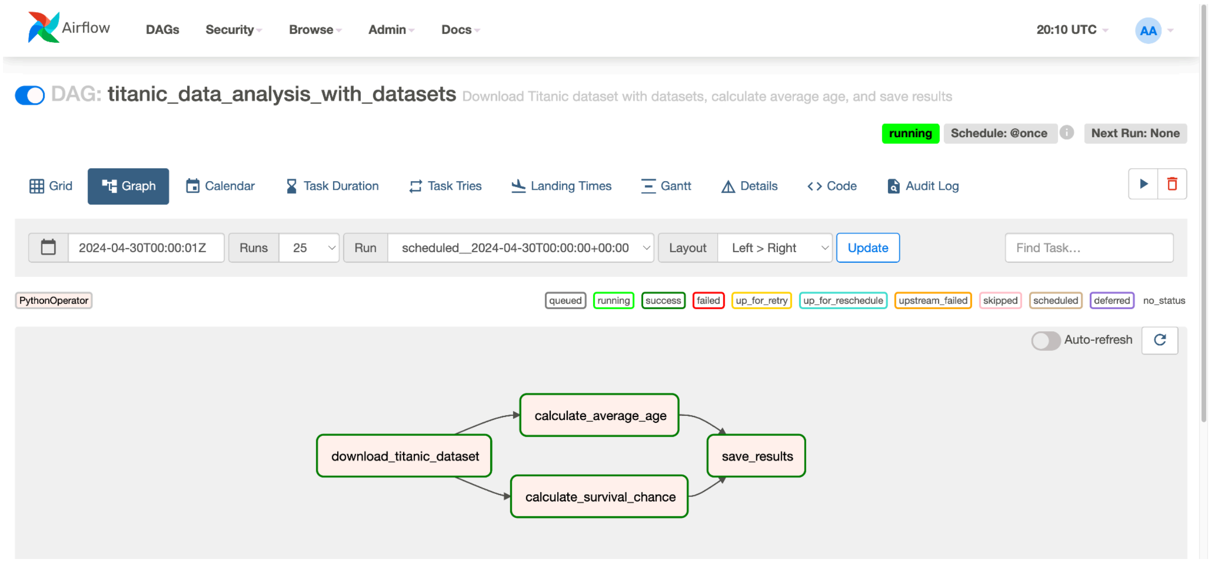
Filter DAGs by tag

Search DAGs

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks
titanic_data_analysis_with_datasets	airflow	1	@once	2024-04-30, 00:00:00		4

Showing 1-1 of 1 DAGs

Triggering DAG via “Trigger DAG” button from actions list

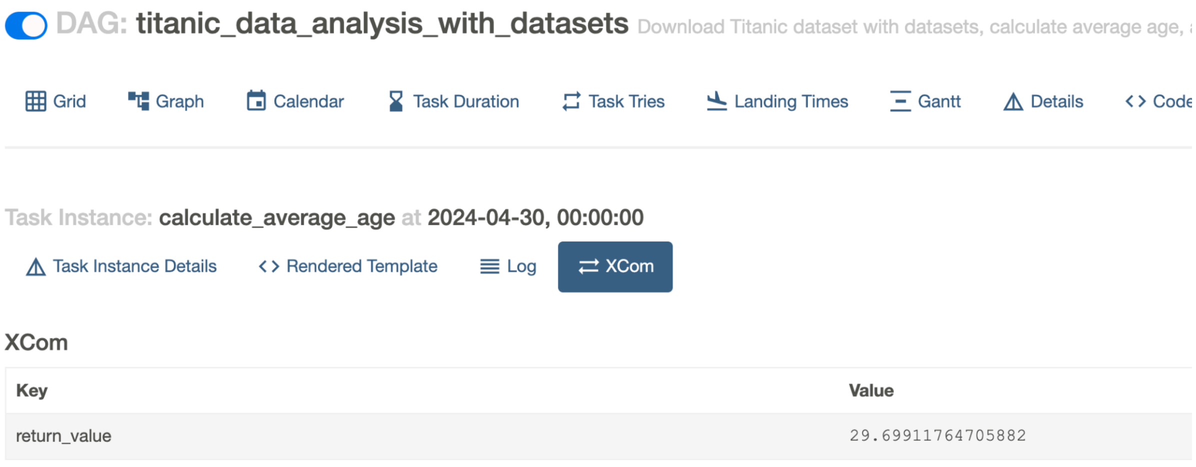


DAG: titanic_data_analysis_with_datasets Download Titanic dataset with datasets, calculate average age, and save results

running | Schedule: @once | Next Run: None

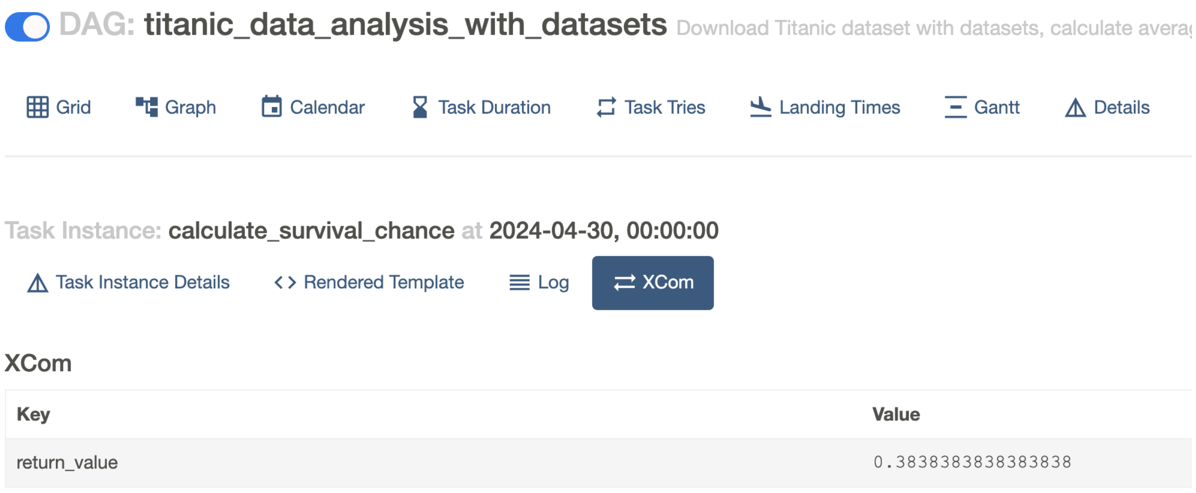
2024-04-30T00:00:01Z | Runs 25 | Run scheduled__2024-04-30T00:00:00+00:00 | Layout Left > Right

And our DAG executed successfully!
We can see tasks execution results as XCOM outputs in interface


DAG: titanic_data_analysis_with_datasets Download Titanic dataset with datasets, calculate average age,
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Task Instance: calculate_average_age at 2024-04-30, 00:00:00
Task Instance Details
Rendered Template
Log
XCom

XCom

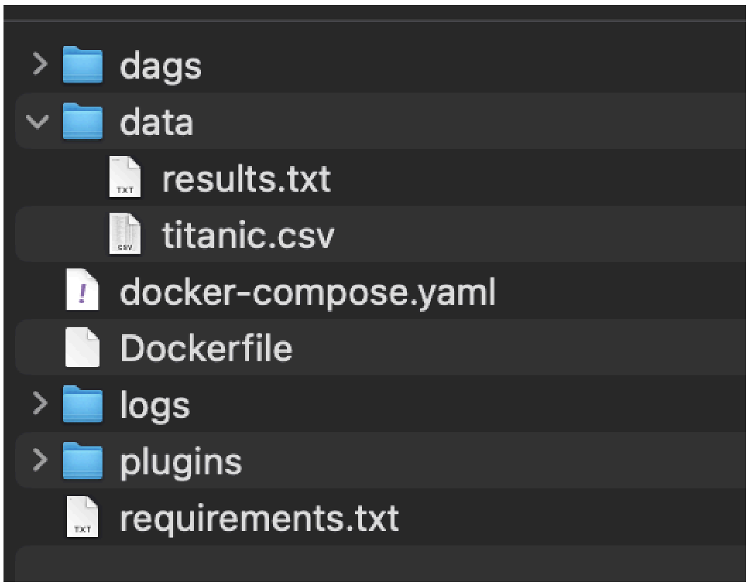
Key	Value
return_value	29.69911764705882

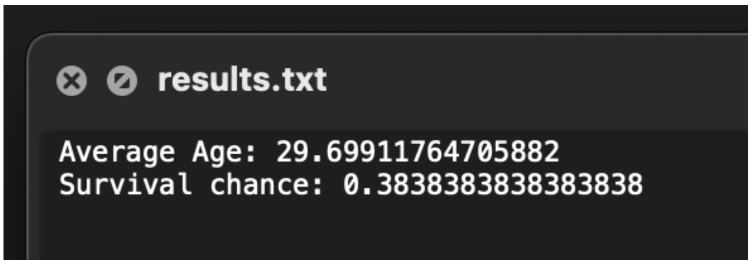

DAG: titanic_data_analysis_with_datasets Download Titanic dataset with datasets, calculate averag
Grid
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Task Instance: calculate_survival_chance at 2024-04-30, 00:00:00
Task Instance Details
Rendered Template
Log
XCom

XCom

Key	Value
return_value	0.3838383838383838

The results are stored in our data folder and contain the calculated metrics:





```
Average Age: 29.69911764705882
Survival chance: 0.3838383838383838
```