Reddit Data Analysis using Apache Airflow and AWS:   
  
  
  
**Services and Tools Used**

**Apache Airflow Services**

* Airflow Scheduler
* Airflow Webserver
* Airflow Workers (Celery based)
* Airflow Metadata Database (PostgreSQL)

**Docker Components**

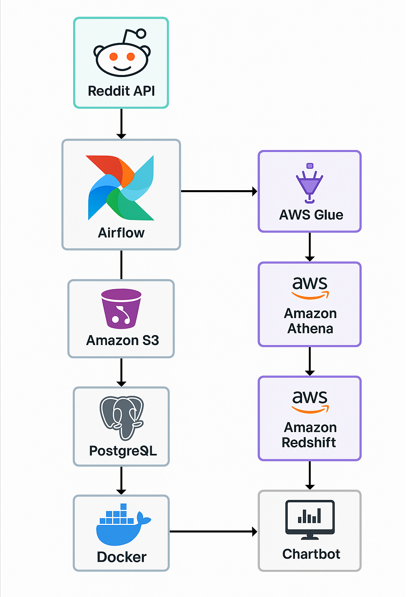
* docker-compose.yml for Airflow cluster
* Dockerfile to build custom Airflow images
* .env files for managing environment variables
* Separate volumes for Airflow logs and database persistence

**AWS Services**

* Amazon S3 (storage of raw Reddit data)
* AWS Glue (crawlers and ETL jobs)
* AWS Athena (SQL querying on S3 data)
* Amazon Redshift (final data warehouse)
* IAM Roles (permissions management)

**Other Tools**

* Chartbot (connected to Redshift for dashboarding)
* Python scripts for Reddit API interaction
* Requirements.txt for Python dependency management
* Configuration files for modular environment management
* PySpark transformations in Glue Jobs

  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
In this project, I designed and implemented a complete end-to-end data pipeline starting from pulling data from Reddit, processing it through Airflow, storing it in AWS S3, and moving it across AWS Glue, Athena, and Redshift, with final visualization using Chartbot connected to Redshift.

First, I created a developer application on Reddit to access its public API. Reddit provided me with a client ID, client secret, and user agent. These keys were essential for authenticating my API requests. I stored these credentials securely inside environment files and Docker secrets so that they would not be hardcoded into the scripts.

I then configured my local Airflow environment. Airflow was backed by a Celery executor setup with a PostgreSQL metadata database. I used Docker Compose to spin up the full environment. I wrote multiple docker-compose.yml files to organize Airflow components like scheduler, webserver, workers, and database containers. I created a requirements.txt file to install additional Python packages needed for Reddit API access and AWS SDK integration.

After setting up Airflow, I created a DAG that handled fetching posts and comments from Reddit using the API. I wrote the DAG in a .py file where I defined tasks to extract data, process it lightly, and push it into an AWS S3 bucket. Airflow was scheduled to run the Reddit ingestion DAG daily. The environment variables, API keys, and configuration paths were loaded dynamically during runtime using a configuration file to keep everything modular and secure.

Once the data was pushed into S3, I connected AWS Glue to the S3 bucket. I configured Glue Crawlers to scan the incoming Reddit data files, infer their schema, and automatically catalog them into the AWS Glue Data Catalog. After that, I used AWS Athena to run SQL queries directly on the S3 data through Glue tables for validation and quick analytics.

Then, I used AWS Glue ETL Jobs to clean, transform, and prepare the Reddit data for loading into Amazon Redshift. Once transformed, I loaded the clean dataset into Redshift tables. Finally, for the visualization layer, I connected Chartbot directly to Redshift and created dashboards showing different Reddit trends, popular subreddits, post frequency, and user engagement.

To manage the code and environments properly, I created multiple environment files for Docker (.env), Airflow configuration files (airflow.cfg), YAML files for AWS services configuration, and separate Python pipeline scripts. I also maintained a requirements.txt file to track all Python dependencies in a clean and reusable manner.

### **Challenges I Faced**

One challenge was securely managing Reddit API keys inside the Docker and Airflow environment without exposing them in code. I solved this by using environment variables and Docker secrets.

Another challenge was ensuring that Airflow tasks that interacted with Reddit and AWS services had the right IAM permissions and network access, especially because I was running Airflow locally inside containers.

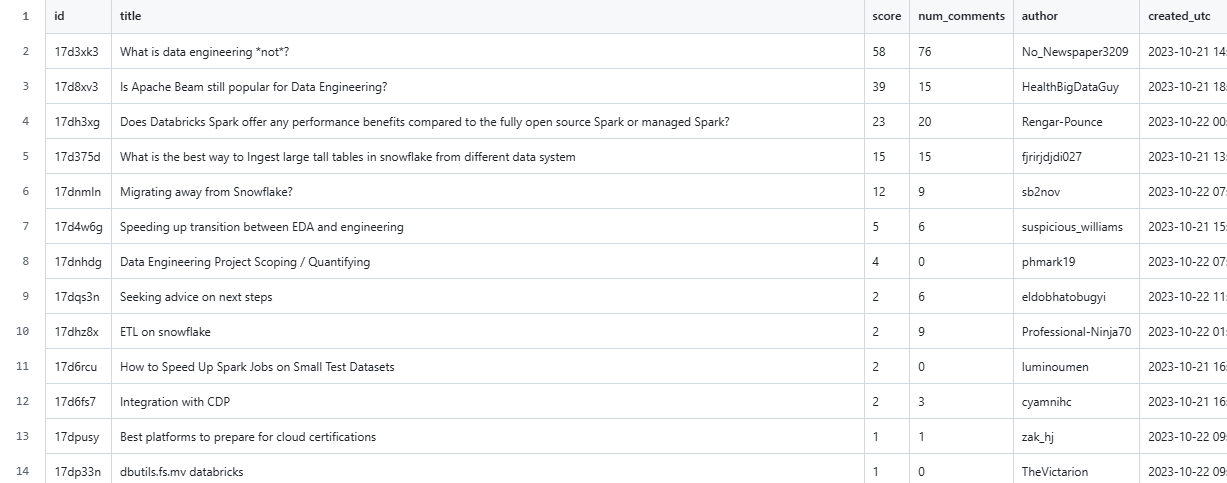
Handling API rate limits and unexpected Reddit API errors was another issue. I added retry logic and error handling inside the Airflow tasks to ensure robustness.

While pushing data into S3, I had to deal with schema evolution, as the Reddit API responses sometimes changed. I wrote dynamic schema inference logic in the Glue Crawler and made my Glue ETL scripts flexible.

There were occasional failures in Redshift loading due to mismatch in field types. I solved this by validating and casting fields properly in the Glue ETL transformation layer before moving to Redshift.

Setting up CeleryExecutor correctly was a bit complex because it required properly configuring broker URLs, result backend settings, and ensuring PostgreSQL stability.

Managing multiple Docker containers together and debugging inter-container communication was another difficulty I faced, especially making sure Airflow webserver, scheduler, and workers could communicate with the Postgres backend and Redis broker properly.

Data:  
  
  
  
Apache Airflow is an open-source platform used to programmatically author, schedule, and monitor workflows. It helps you automate and manage complex data pipelines by organizing tasks and running them in a defined order.

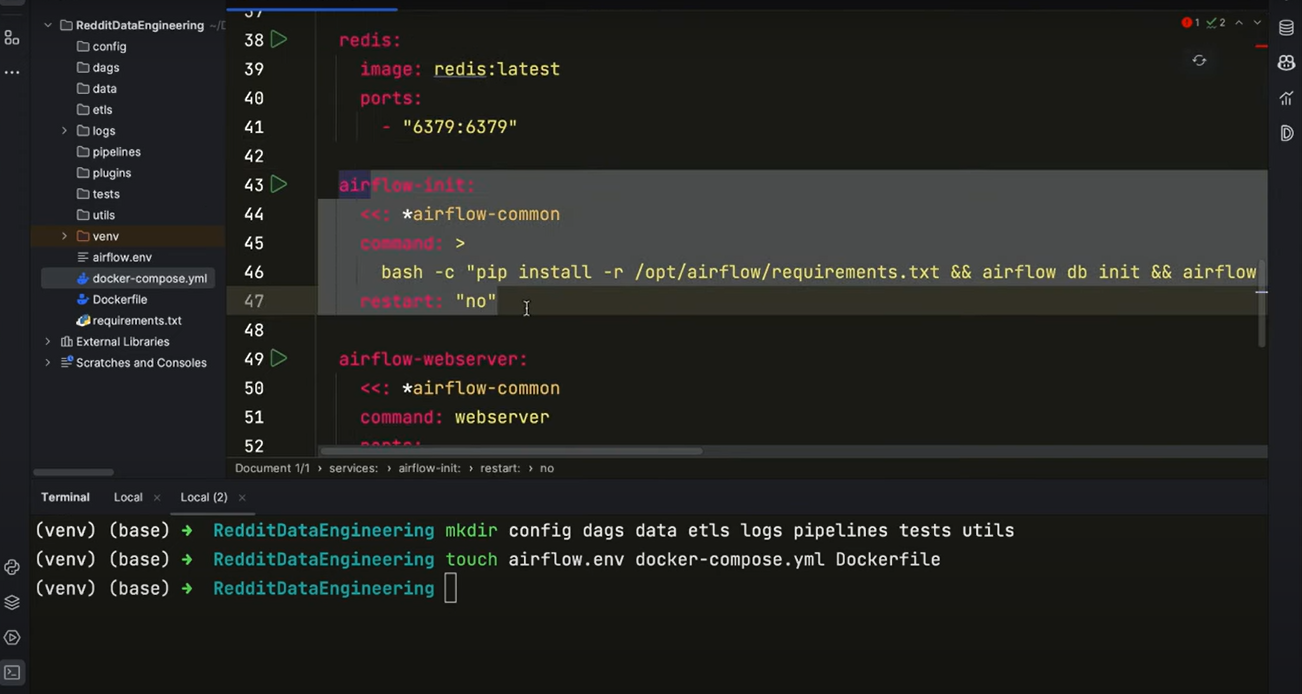
In Airflow, workflows are defined as Directed Acyclic Graphs, also called DAGs. Each node in the DAG is a task, and the edges show the dependencies between tasks. Airflow makes sure tasks are executed in the right sequence based on these dependencies.

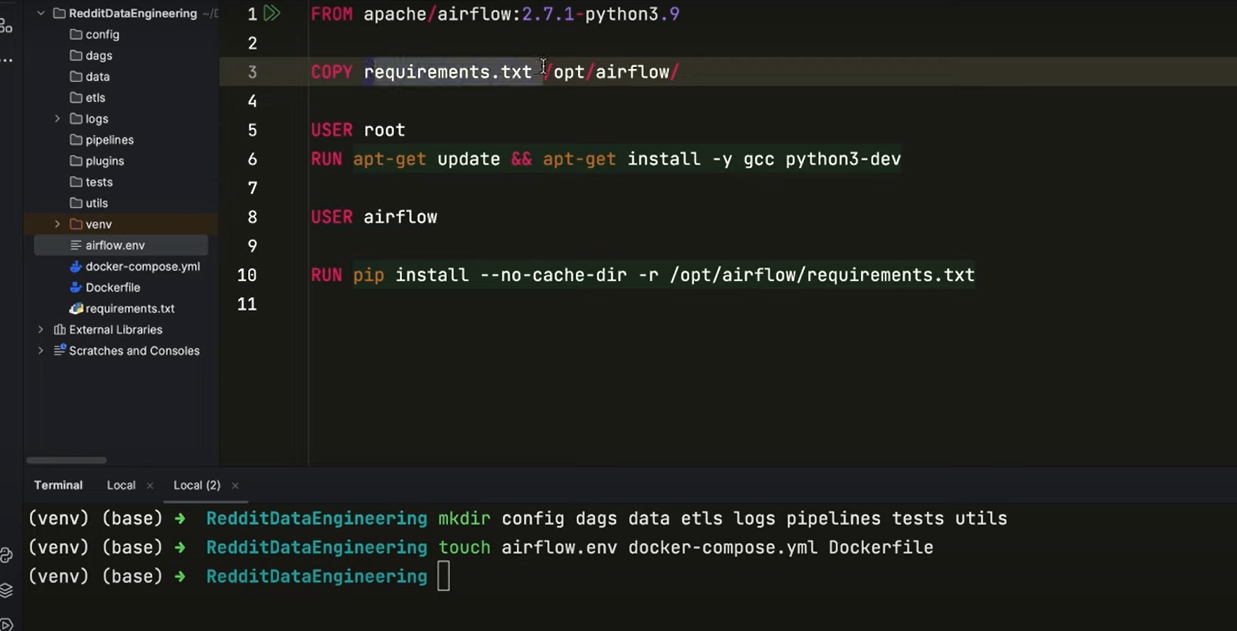
You write workflows in Python code, which makes it flexible and powerful. You can define what needs to happen, how often it should run, what should happen if a task fails, and many other conditions.

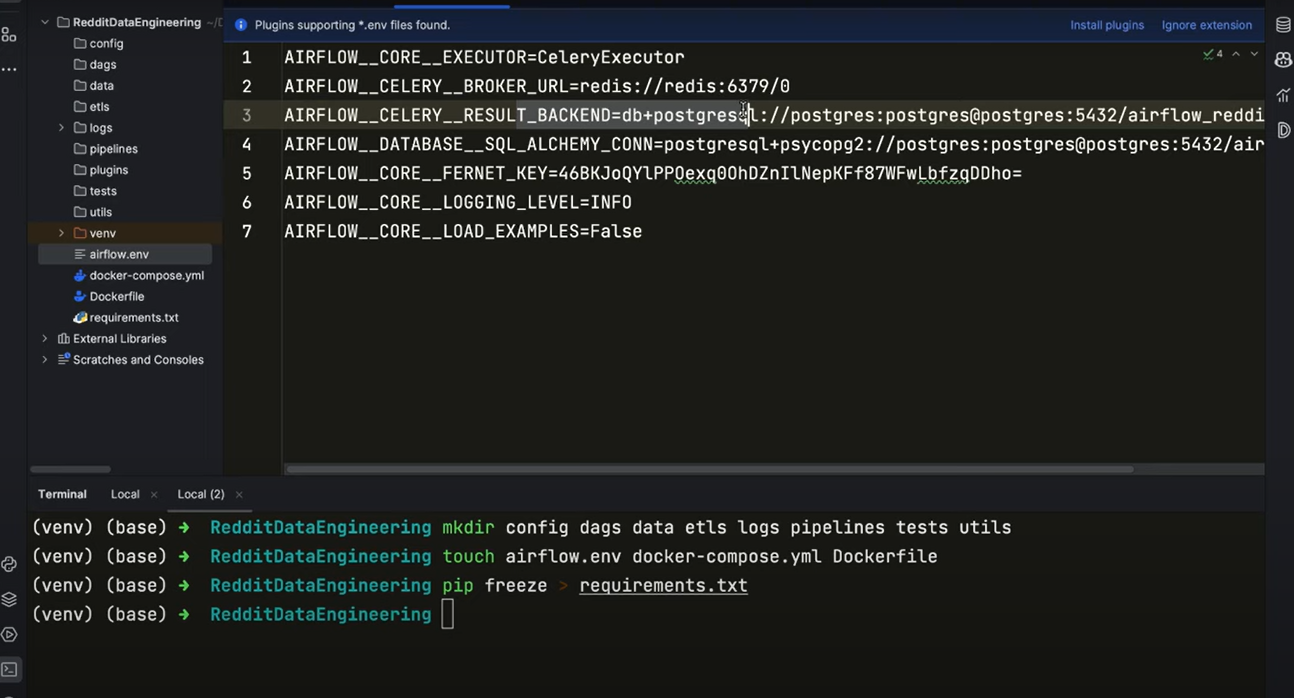
Airflow has a built-in scheduler that triggers tasks based on time or external events. It also provides a web-based UI where you can see the status of your workflows, check logs, retry failed tasks, and monitor the progress of your pipelines.

Airflow is widely used for data engineering, ETL jobs, machine learning pipelines, cloud automation, and anything where multiple steps need to happen in order and be tracked reliably.

1. Configuring Apache Airflow:



Docker file setup: for security  


Airflow env:  
  
  
  
Reddit pipeline.py  
import pandas as pd

from etls.reddit\_etl import connect\_reddit, extract\_posts, transform\_data, load\_data\_to\_csv

from utils.constants import CLIENT\_ID, SECRET, OUTPUT\_PATH

def reddit\_pipeline(file\_name: str, subreddit: str, time\_filter='day', limit=None):

# connecting to reddit instance

instance = connect\_reddit(CLIENT\_ID, SECRET, 'Airscholar Agent')

# extraction

posts = extract\_posts(instance, subreddit, time\_filter, limit)

post\_df = pd.DataFrame(posts)

# transformation

post\_df = transform\_data(post\_df)

# loading to csv

file\_path = f'{OUTPUT\_PATH}/{file\_name}.csv'

load\_data\_to\_csv(post\_df, file\_path)

return file\_path  
  
  
AWS S3 pipeline:  
  
from etls.aws\_etl import connect\_to\_s3, create\_bucket\_if\_not\_exist, upload\_to\_s3

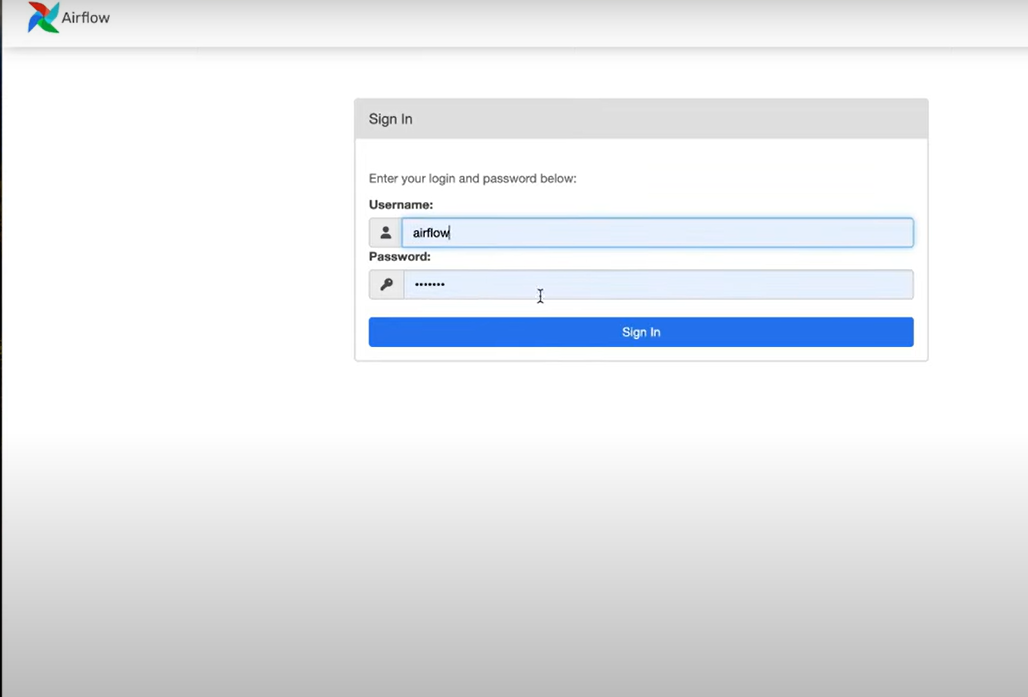
from utils.constants import AWS\_BUCKET\_NAME

def upload\_s3\_pipeline(ti):

file\_path = ti.xcom\_pull(task\_ids='reddit\_extraction', key='return\_value')

s3 = connect\_to\_s3()

create\_bucket\_if\_not\_exist(s3, AWS\_BUCKET\_NAME)

upload\_to\_s3(s3, file\_path, AWS\_BUCKET\_NAME, file\_path.split('/')[-1])  
  
  
  
Airflow login:  
  
  
  
redditetl.py  
import sys

import numpy as np

import pandas as pd

import praw

from praw import Reddit

from utils.constants import POST\_FIELDS

def connect\_reddit(client\_id, client\_secret, user\_agent) -> Reddit:

try:

reddit = praw.Reddit(client\_id=client\_id,

client\_secret=client\_secret,

user\_agent=user\_agent)

print("connected to reddit!")

return reddit

except Exception as e:

print(e)

sys.exit(1)

def extract\_posts(reddit\_instance: Reddit, subreddit: str, time\_filter: str, limit=None):

subreddit = reddit\_instance.subreddit(subreddit)

posts = subreddit.top(time\_filter=time\_filter, limit=limit)

post\_lists = []

for post in posts:

post\_dict = vars(post)

post = {key: post\_dict[key] for key in POST\_FIELDS}

post\_lists.append(post)

return post\_lists

def transform\_data(post\_df: pd.DataFrame):

post\_df['created\_utc'] = pd.to\_datetime(post\_df['created\_utc'], unit='s')

post\_df['over\_18'] = np.where((post\_df['over\_18'] == True), True, False)

post\_df['author'] = post\_df['author'].astype(str)

edited\_mode = post\_df['edited'].mode()

post\_df['edited'] = np.where(post\_df['edited'].isin([True, False]),

post\_df['edited'], edited\_mode).astype(bool)

post\_df['num\_comments'] = post\_df['num\_comments'].astype(int)

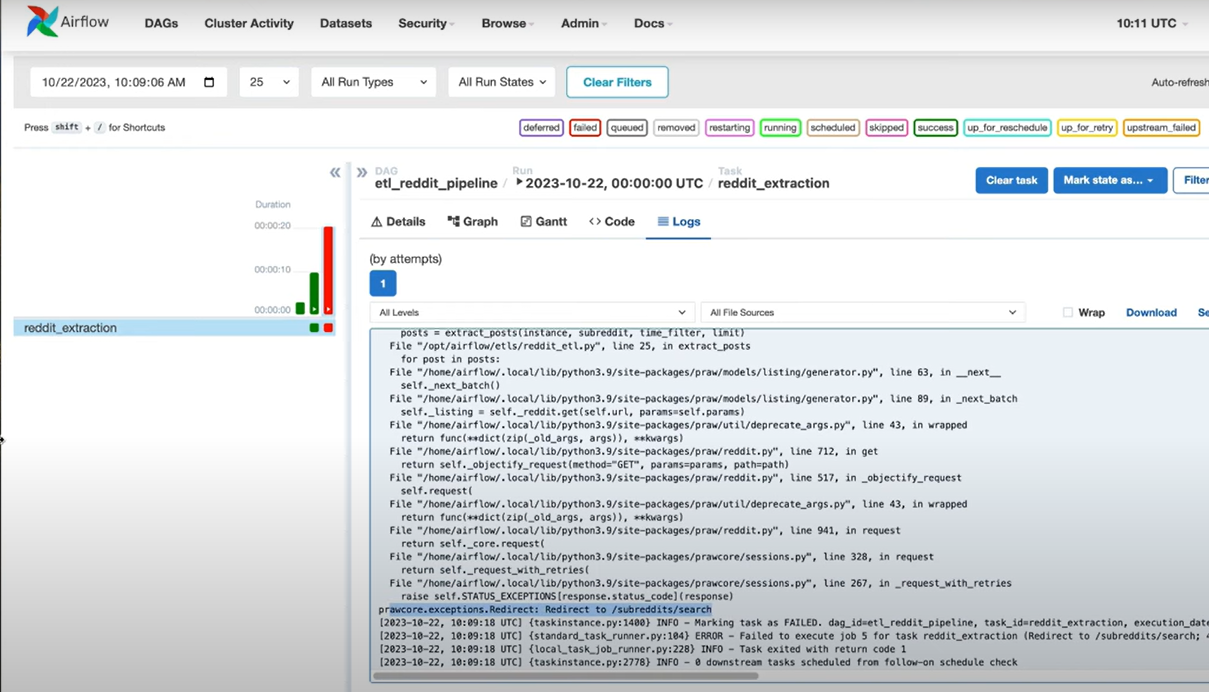
post\_df['score'] = post\_df['score'].astype(int)

post\_df['title'] = post\_df['title'].astype(str)

return post\_df

def load\_data\_to\_csv(data: pd.DataFrame, path: str):

data.to\_csv(path, index=False)

  
  
Constants.py  
  
import configparser

import os

parser = configparser.ConfigParser()

parser.read(os.path.join(os.path.dirname(\_\_file\_\_), '../config/config.conf'))

SECRET = parser.get('api\_keys', 'reddit\_secret\_key')

CLIENT\_ID = parser.get('api\_keys', 'reddit\_client\_id')

DATABASE\_HOST = parser.get('database', 'database\_host')

DATABASE\_NAME = parser.get('database', 'database\_name')

DATABASE\_PORT = parser.get('database', 'database\_port')

DATABASE\_USER = parser.get('database', 'database\_username')

DATABASE\_PASSWORD = parser.get('database', 'database\_password')

#AWS

AWS\_ACCESS\_KEY\_ID = parser.get('aws', 'aws\_access\_key\_id')

AWS\_ACCESS\_KEY = parser.get('aws', 'aws\_secret\_access\_key')

AWS\_REGION = parser.get('aws', 'aws\_region')

AWS\_BUCKET\_NAME = parser.get('aws', 'aws\_bucket\_name')

INPUT\_PATH = parser.get('file\_paths', 'input\_path')

OUTPUT\_PATH = parser.get('file\_paths', 'output\_path')

POST\_FIELDS = (

'id',

'title',

'score',

'num\_comments',

'author',

'created\_utc',

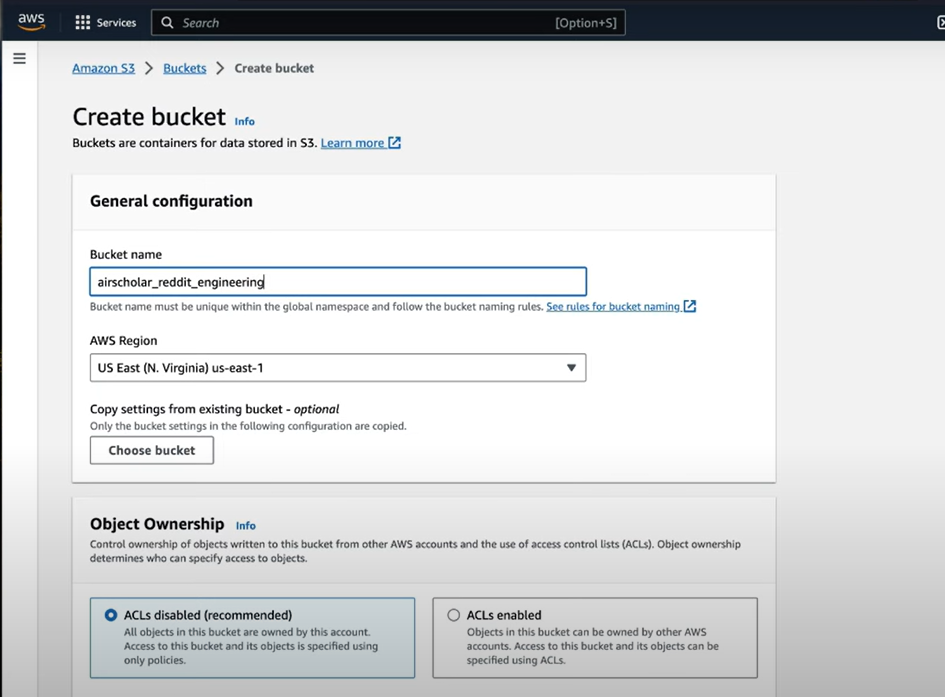
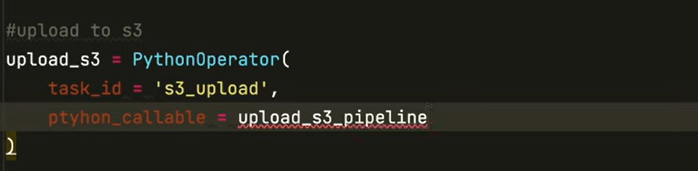
'url',

'over\_18',

'edited',

'spoiler',

'stickied'

)  
  
  
Configure:  
AWS CLI:  
  
  
Creation of s3 bucket:  
  
  
  
Upload to S3:  
  
  
Reddit\_dag.py  
  
import os

import sys

from datetime import datetime

from airflow import DAG

from airflow.operators.python import PythonOperator

sys.path.insert(0, os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_))))

from pipelines.aws\_s3\_pipeline import upload\_s3\_pipeline

from pipelines.reddit\_pipeline import reddit\_pipeline

default\_args = {

'owner': 'Yusuf Ganiyu',

'start\_date': datetime(2023, 10, 22)

}

file\_postfix = datetime.now().strftime("%Y%m%d")

dag = DAG(

dag\_id='etl\_reddit\_pipeline',

default\_args=default\_args,

schedule\_interval='@daily',

catchup=False,

tags=['reddit', 'etl', 'pipeline']

)

# extraction from reddit

extract = PythonOperator(

task\_id='reddit\_extraction',

python\_callable=reddit\_pipeline,

op\_kwargs={

'file\_name': f'reddit\_{file\_postfix}',

'subreddit': 'dataengineering',

'time\_filter': 'day',

'limit': 100

},

dag=dag

)

# upload to s3

upload\_s3 = PythonOperator(

task\_id='s3\_upload',

python\_callable=upload\_s3\_pipeline,

dag=dag

)

extract >> upload\_s3  
  
  
  
Aws\_etl.py  
import s3fs

from utils.constants import AWS\_ACCESS\_KEY\_ID, AWS\_ACCESS\_KEY

def connect\_to\_s3():

try:

s3 = s3fs.S3FileSystem(anon=False,

key= AWS\_ACCESS\_KEY\_ID,

secret=AWS\_ACCESS\_KEY)

return s3

except Exception as e:

print(e)

def create\_bucket\_if\_not\_exist(s3: s3fs.S3FileSystem, bucket:str):

try:

if not s3.exists(bucket):

s3.mkdir(bucket)

print("Bucket created")

else :

print("Bucket already exists")

except Exception as e:

print(e)

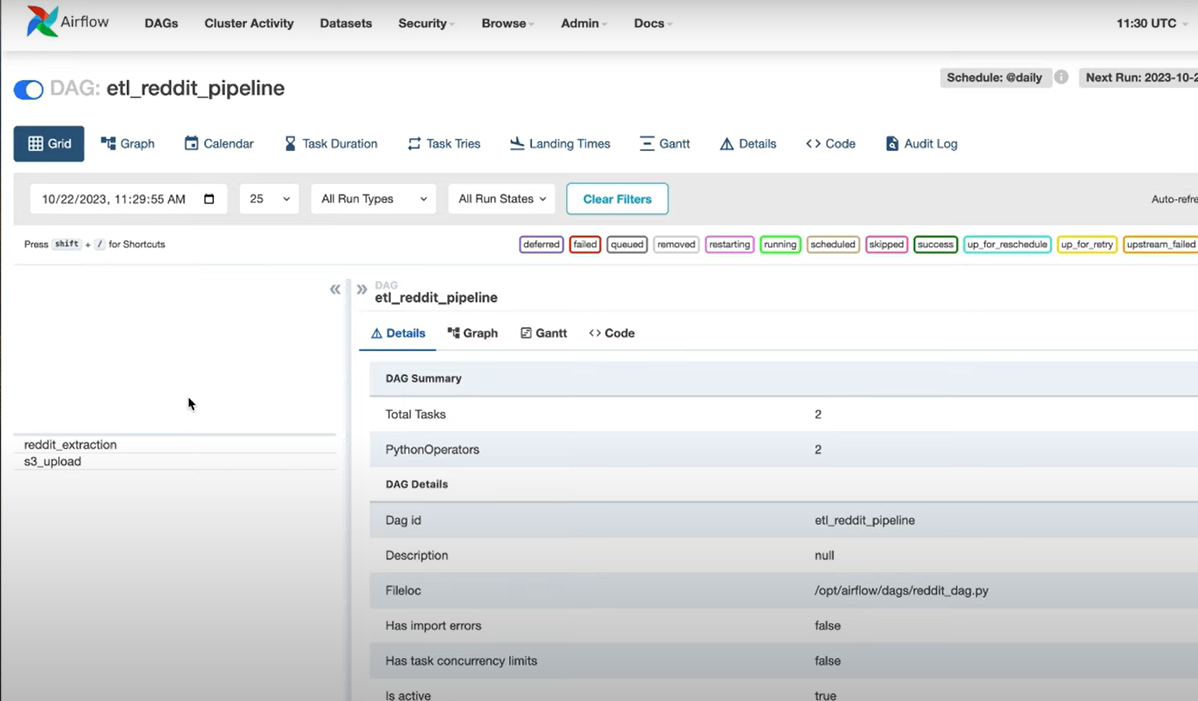
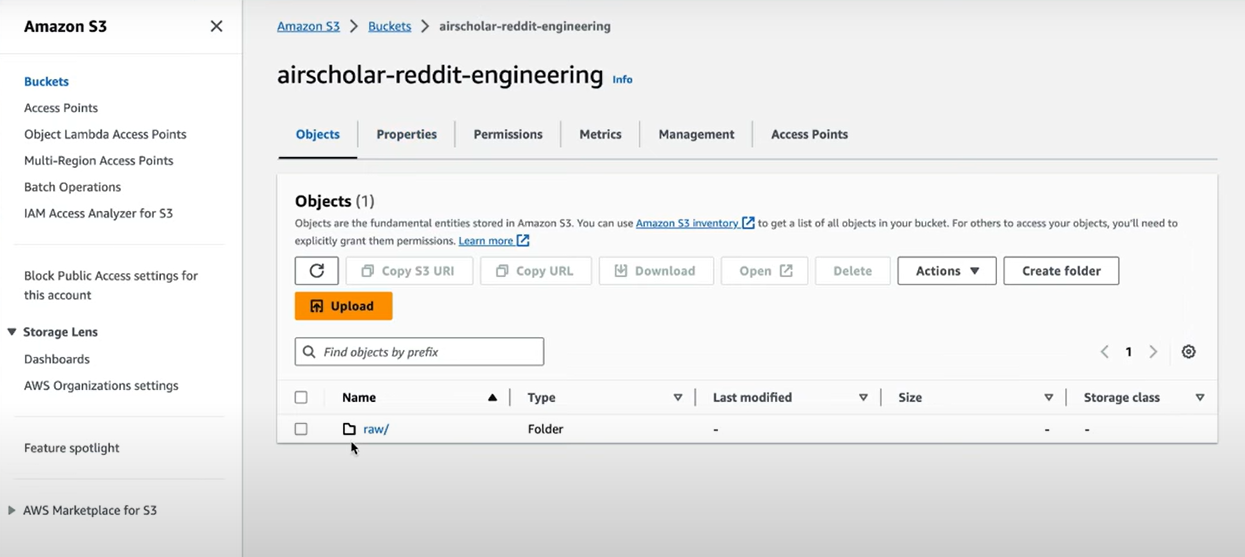
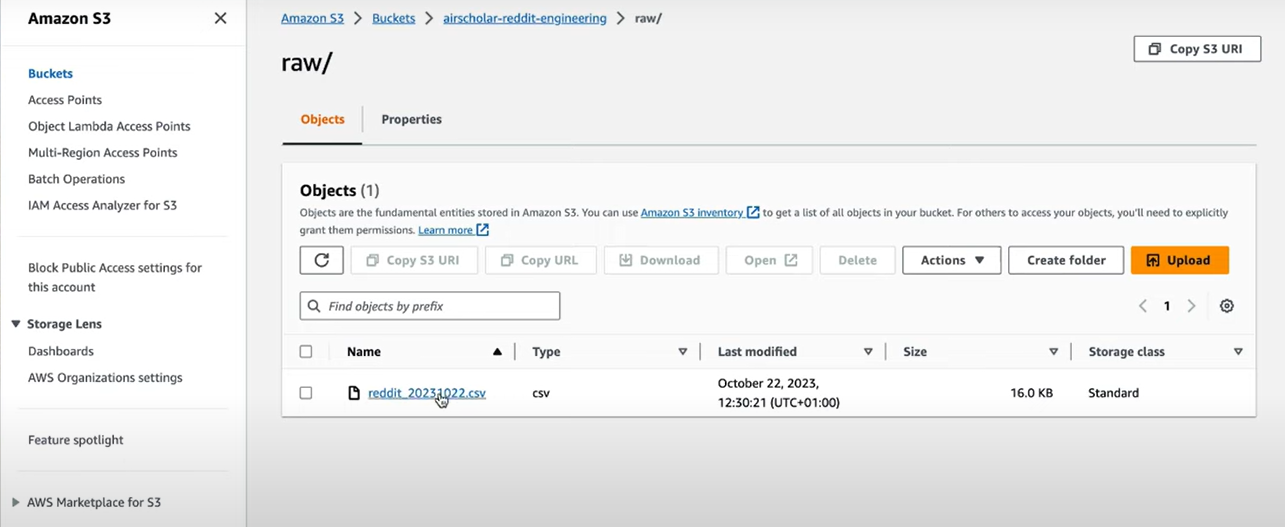
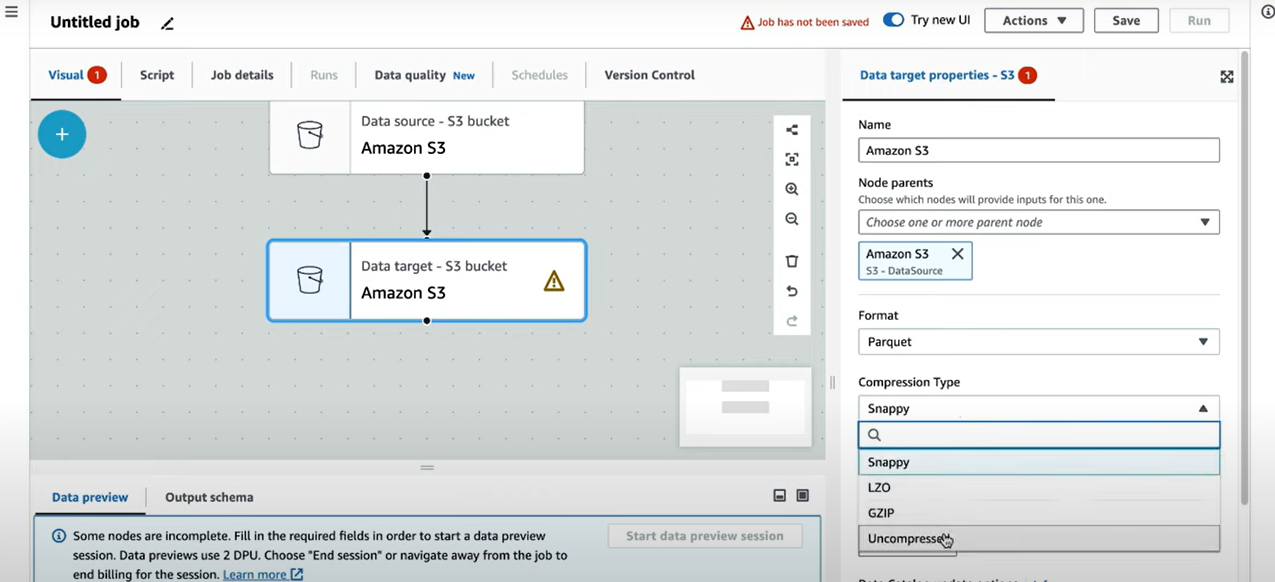
def upload\_to\_s3(s3: s3fs.S3FileSystem, file\_path: str, bucket:str, s3\_file\_name: str):

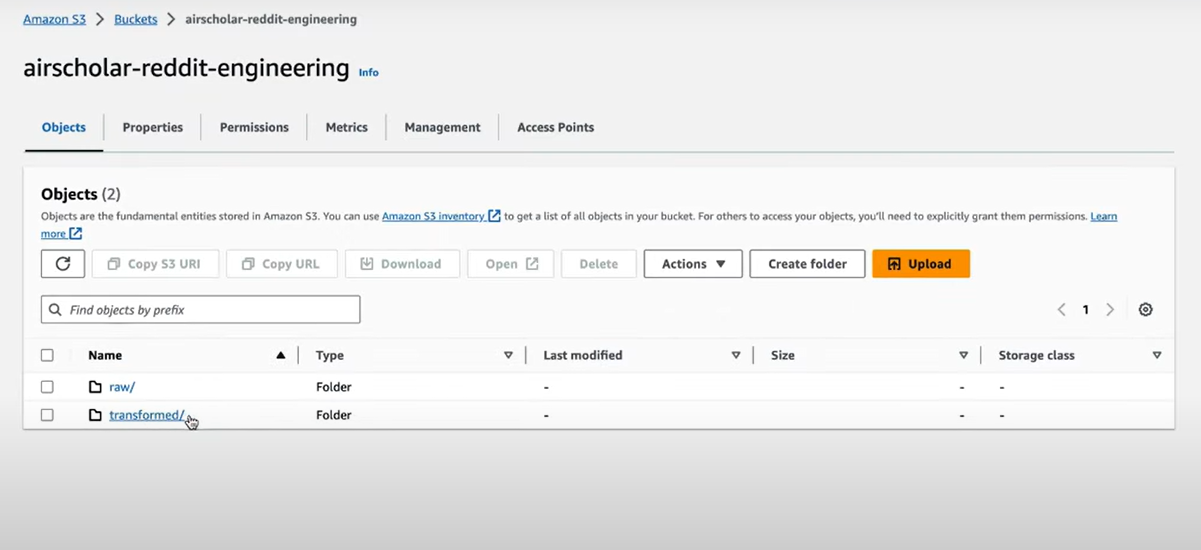
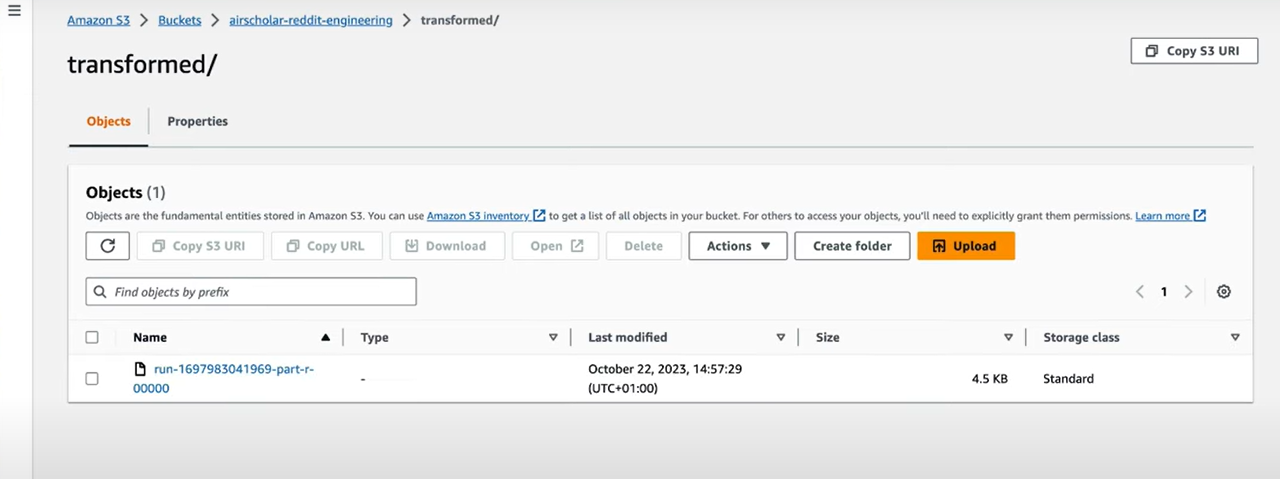
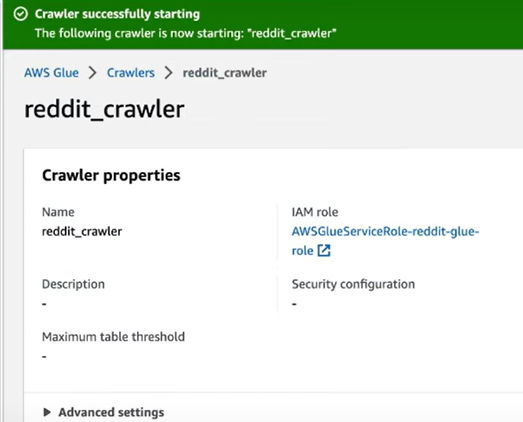
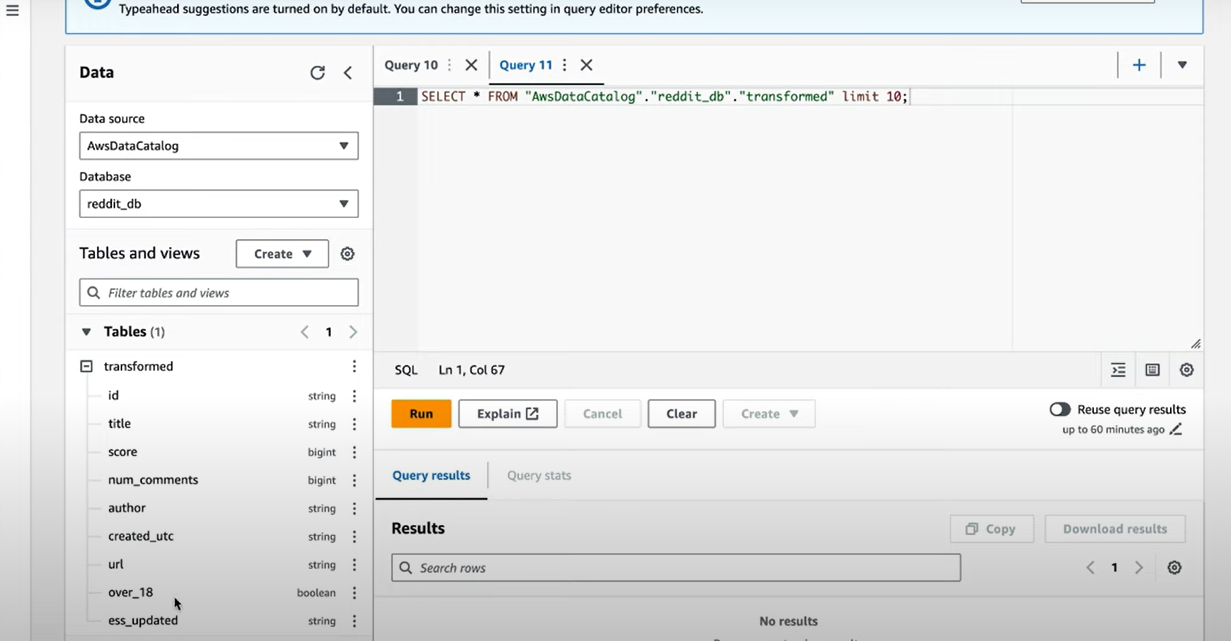
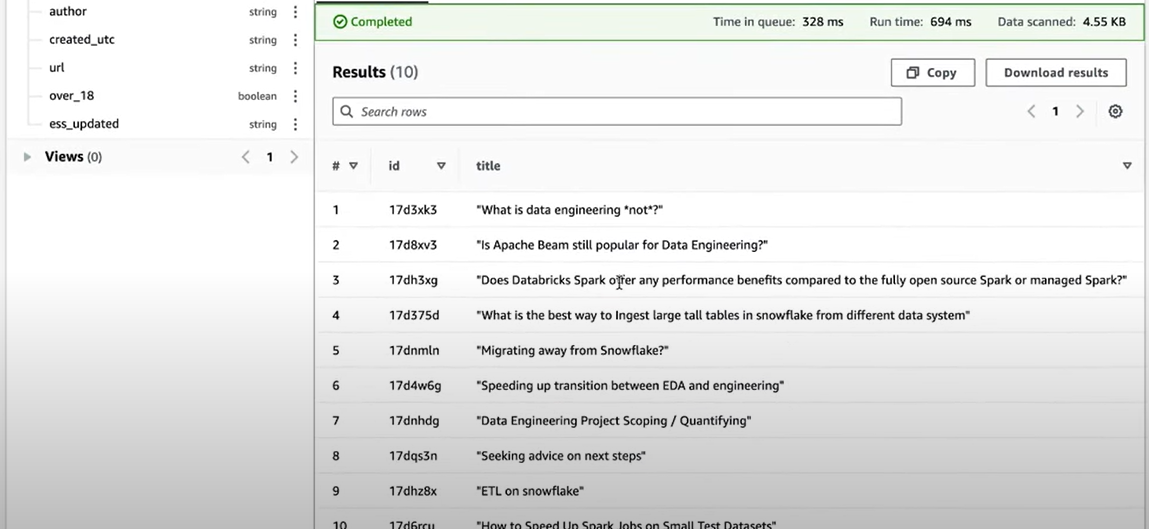
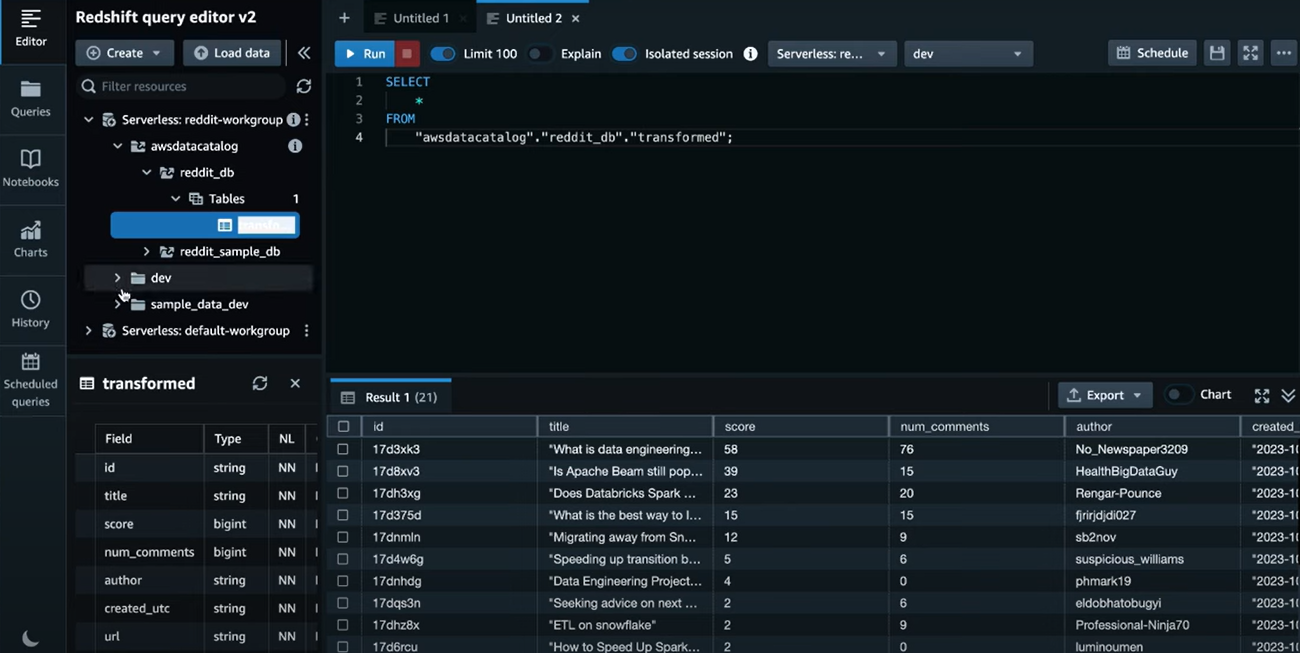
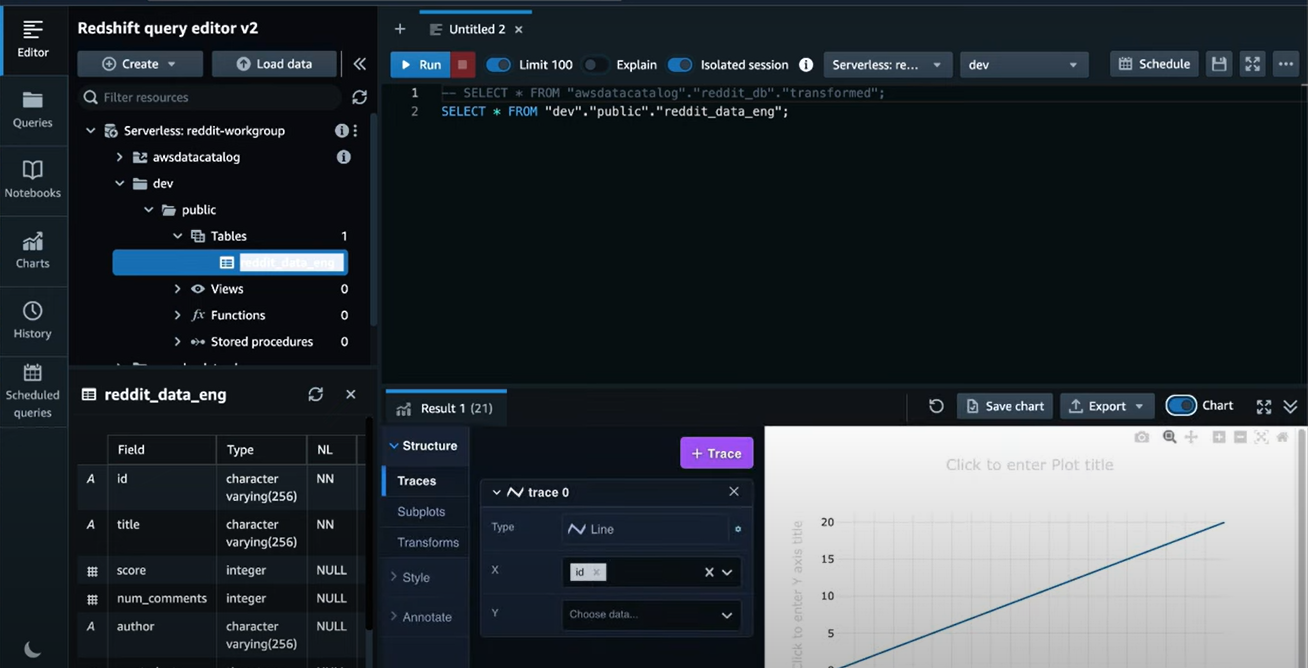
try:

s3.put(file\_path, bucket+'/raw/'+ s3\_file\_name)

print('File uploaded to s3')

except FileNotFoundError:

print('The file was not found')  
  
  
  
  
Just enable the pipeline from Apache Airflow.  
  
Data pushed to s3 from Apache Airflow:  
  
  
  
  
  
GLue ETL Job:  
  
  
  
Transformed folder in S3 for glue output:

  
  
Some scripted transformations in glue job as well.  
  
  
  
  
  
  
  
  
After running glue job, I see the parquet file in transformed s3.  
  
  
  
Create Crawler: input as created Parquet:  
  
  
  
 in S3 create transform Datina as a bucket for querying the data for the output which the crawler generated from Athena.  
  
  
  
  
  
  
  
  
  
Now lets configure redshift.  
  
  
  
  
 Enable chart in red set for viewing the query data or transform data  
  
.  
  
  
