Data Lake Project Starter Code

The objective of this project is to apply what we've learned on Spark and data lakes to build an ETL pipeline for a data lake hosted on S3.

How to run the program with your own code

For the execution of your own code, we head to the Project Workspace.

Running the etl.py script in a terminal to process the all the datasets. The script connects to the Sparkify database, extracts and processes the log_data and song_data, and loads data into the five tables.

```
python etl.py
```

The summary of the files and folders within repo is provided in the table below:

File/Folder	Definition

File/Folder	Definition
data/*	Folder that contains all the json files with the data used in this project.
images/*	Folder containing the images of the project.
dl.cfg	Contains our AWS credentials.
etl.py	Contains all the processing of reads data from S3, processes that data using Spark, and writes them back to S3.
README.md	Contains the project documentation.
README.pdf	Contains the project documentation in PDF format.

Steps to complete the project:

1. To complete the project, you will need to load data from S3, process the data into analytics tables using Spark, and load them back into S3. You'll deploy this Spark process on a cluster using AWS..

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Scenario.

A music streaming startup, Sparkify, has grown their user base and song database even more and want to move their data warehouse to a data lake. Their data resides in S3, in a directory of JSON logs on user activity on the app, as well as a directory with JSON metadata on the songs in their app.

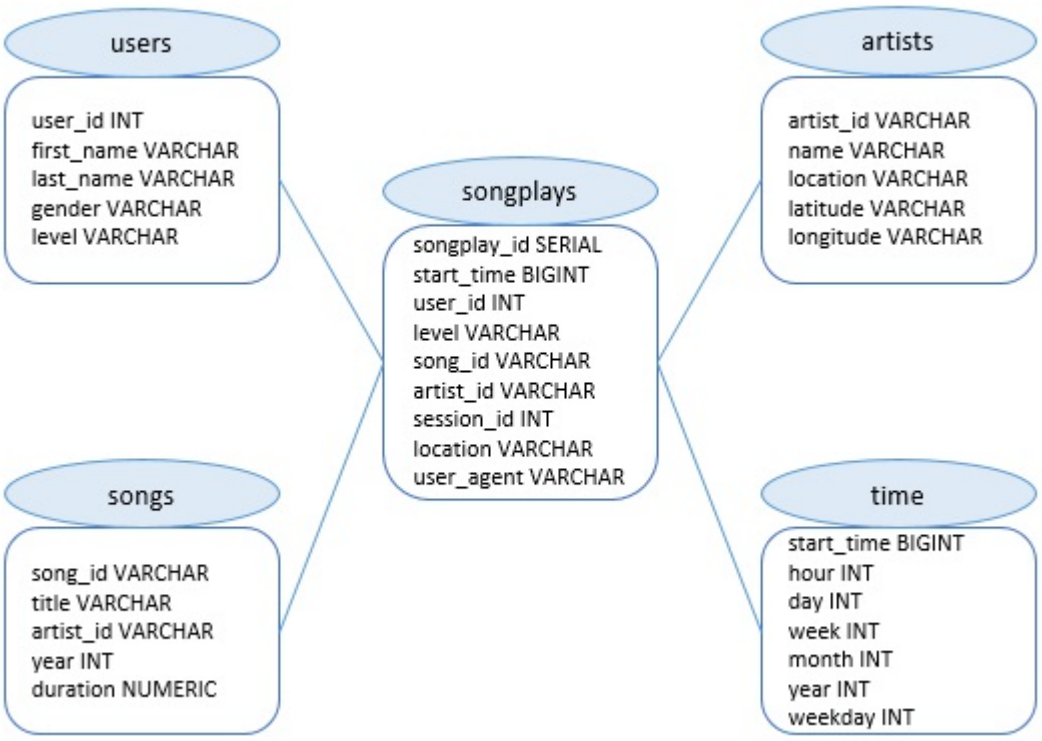
As their data engineer, you are tasked with building an ETL pipeline that extracts their data from S3, processes them using Spark, and loads the data back into S3 as a set of dimensional tables. This will allow their analytics team to continue finding insights in what songs their users are listening to.

You'll be able to test your database and ETL pipeline by running queries given to you by the analytics team from Sparkify and compare your results with their expected results.

Database Schema for Sparkify.

In our scenario, we have one dataset of songs where each file is in JSON format and contains metadata about a song and the artist of that song. The second dataset consists of log files in JSON format and simulates activity logs from a music streaming app (based on specified configurations).

Using the song and log datasets, the database schema looks like this:



Star Schema.

A star schema is the simplest style of data mart schema. The star schema consists of one or more fact tables referencing to any number of dimension tables. It has some advantages like fast aggregation for analytics, simple queries for JOINs, etc.

ETL Pipeline

Extract, transform, load (ETL) is the general procedure of copying data from one or more sources into a destination system which represents the data differently from, or in a different context than, the sources.

Project Datasets

We'll be working with two datasets that reside in S3. Here are the S3 links for each:

- Song data: `s3://udacity-dend/song_data`
- Log data: `s3://udacity-dend/log_data`

Song Dataset

The first dataset is a subset of real data from the Million Song Dataset. Each file is in JSON format and contains metadata about a song and the artist of that song. The files are partitioned by the first three letters of each song's track ID. For example, here are filepaths to three files in this dataset.

```
song_data/A/A/A/TRAAAAW128F429D538.json
song_data/A/A/B/TRAABCL128F4286650.json
song_data/A/A/C/TRAACCG128F92E8A55.json
```

And below is an example of what a single song file, TRAAAAW128F429D538.json, looks like.


```
{"num_songs": 1, "artist_id": "ARD7TVE1187B99BFB1", "artist_latitude": null, "artist
```


Log Dataset

The second dataset consists of log files in JSON format generated by this event simulator based on the songs in the dataset above. These simulate app activity logs from an imaginary music streaming app based on configuration settings.

The log files in the dataset we'll be working with are partitioned by year and month. For example, here are filepaths to three files in this dataset.

```
log_data/2018/11/2018-11-01-events.json
log_data/2018/11/2018-11-02-events.json
log_data/2018/11/2018-11-03-events.json
```

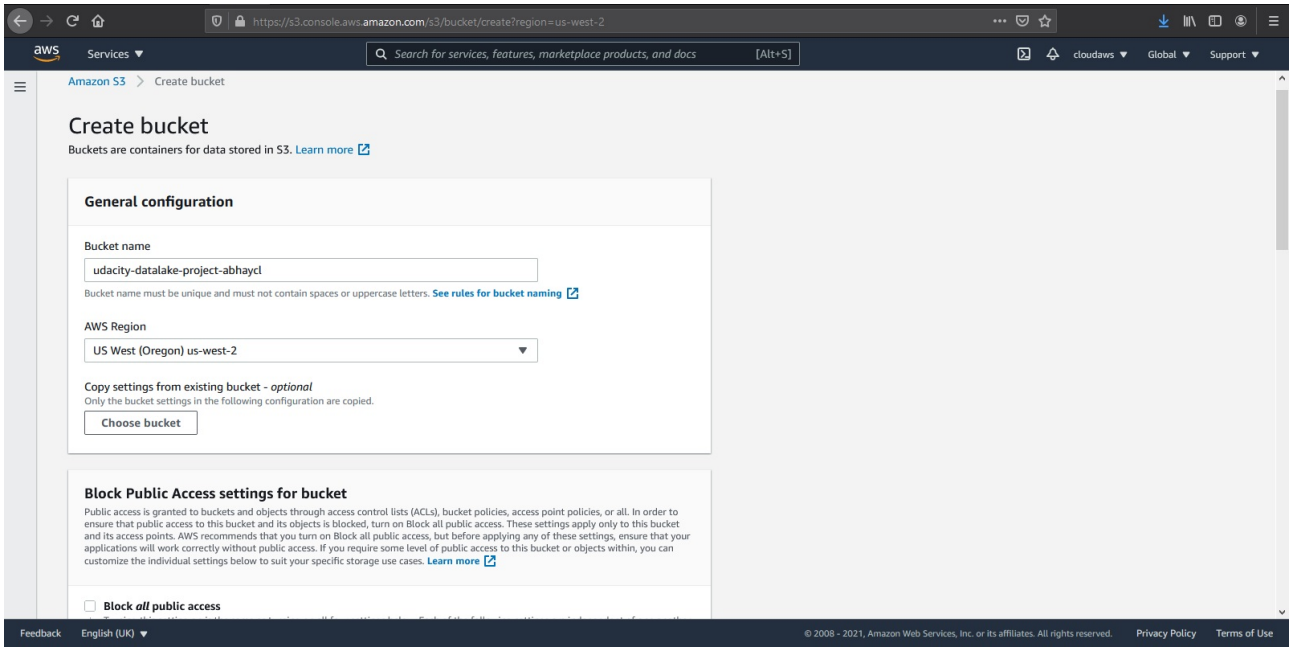
And below is an example of what the data in a log file, *2018-11-01-events.json*, looks like.

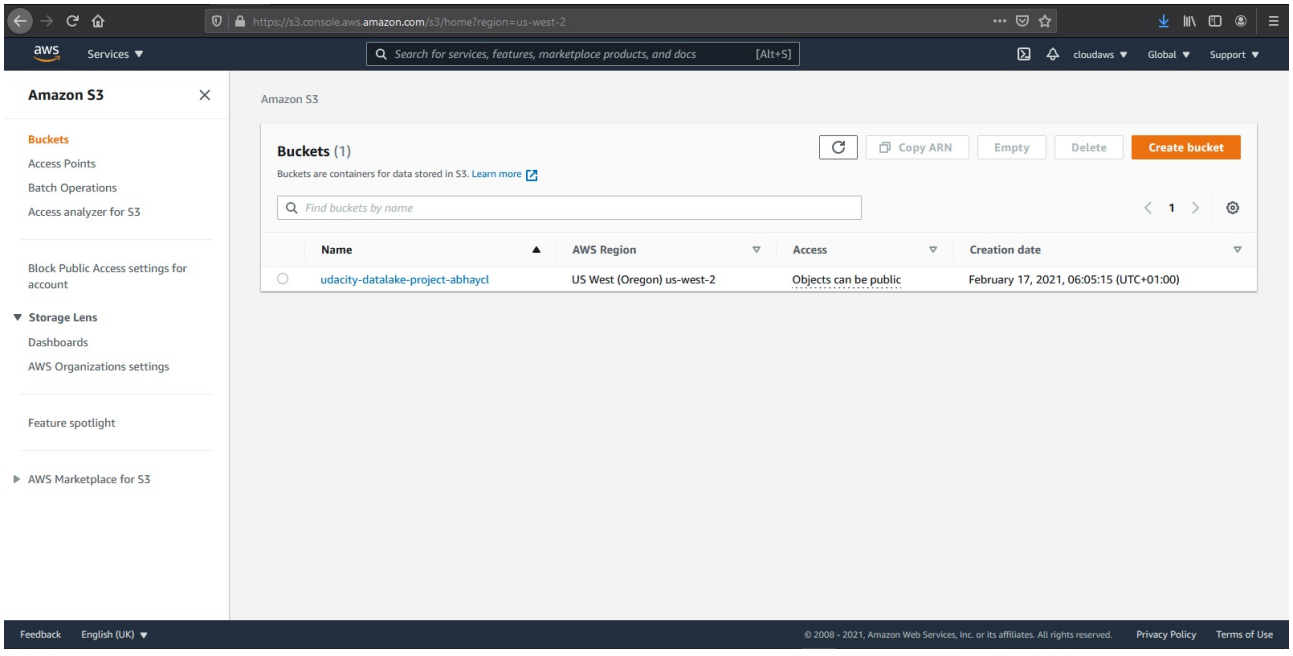

```
{"artist":null,"auth":"Logged In","firstName":"Walter","gender":"M","itemInSession":
```

Prerequisites

It is necessary that we have an AWS account. The following tasks must be realized within AWS:

1. In the IAM area you have to create a user with read and write permissions or full access to the S3 buckets area.
2. In the S3 area you have to create an S3 bucket, in our case we call it udacity-datalake-project-abhaycl.





Possible help commands from the console:

- Empty the content of the S3 bucket: `aws s3 rm s3://udacity-datalake-project-abhaycl --recursive`
- Delete the previously created S3 bucket: `aws s3 rb s3://udacity-datalake-project-abhaycl --force`

Note: If we have an AWS Free Tier account, each month only allows 2000 Put, Copy, Post or List Requests of Amazon S3, this may limit the reading of the song data files from the source S3 bucket (s3a://udacity-dend/song_data/) on our destination S3 bucket (s3://udacity-datalake-project-abhaycl).

Process song data (song_data directory).

We will perform ETL on the files in song_data directory to create two dimensional tables: songs table and artists table.

This is what a songs file looks like:

```
{"num_songs": 1, "artist_id": "ARD7TVE1187B99BFB1", "artist_latitude": null, "artist
```

We will proceed to read the data from the source S3 bucket, in the image below we can check the result of the task.

```
df = spark.read.json(song_data)
```

```
--- Reading song data from json files. --- 2021-02-17
- Shows the scheme and some song data records...
root
 |-- artist_id: string (nullable = true)
 |-- artist_latitude: double (nullable = true)
 |-- artist_location: string (nullable = true)
 |-- artist_longitude: double (nullable = true)
 |-- artist_name: string (nullable = true)
 |-- duration: double (nullable = true)
 |-- num_songs: long (nullable = true)
 |-- song_id: string (nullable = true)
 |-- title: string (nullable = true)
 |-- year: long (nullable = true)

+------------------+---------------+--------------------+----------------+--------------------+---------+---------+------------------+--------------------+----+
|         artist_id|artist_latitude|     artist_location|artist_longitude|         artist_name| duration|num_songs|           song_id|               title|year|
+------------------+---------------+--------------------+----------------+--------------------+---------+---------+------------------+--------------------+----+
|ARTC1LV1187B9A4858|        51.4536|Goldsmith's Colle...|        -0.01802|  The Bonzo Dog Band|301.40036|        1|SOAFBCP12A8C13CC7D|King Of Scurf (20...|1972|
|ARA23XO1187B9AF18F|       40.57885|Carteret, New Jersey|       -74.21956|     The Smithereens|  192.522|        1|SOKTJDS12AF72A25E5|Drown In My Own T...|   0|
|AR5VTNL1187B992A91|       51.50632|     London, England|        -0.12714|       Jonathan King|129.85424|        1|SOEKAZG12AB018837E|I'll Slap Your Fa...|2001|
|AR73AIO1187B9AD57B|       37.77916|   San Francisco, CA|      -122.42005|   Western Addiction|118.07302|        1|SOQPWCR12A6D4FB2A3|A Poor Recipe For...|2005|
|ARXQBR11187B98A2CC|           null|  Liverpool, England|            null|Frankie Goes To H...|821.05424|        1|SOBRKGM12A8C139EF6|Welcome to the Pl...|1985|
+------------------+---------------+--------------------+----------------+--------------------+---------+---------+------------------+--------------------+----+
only showing top 5 rows
```

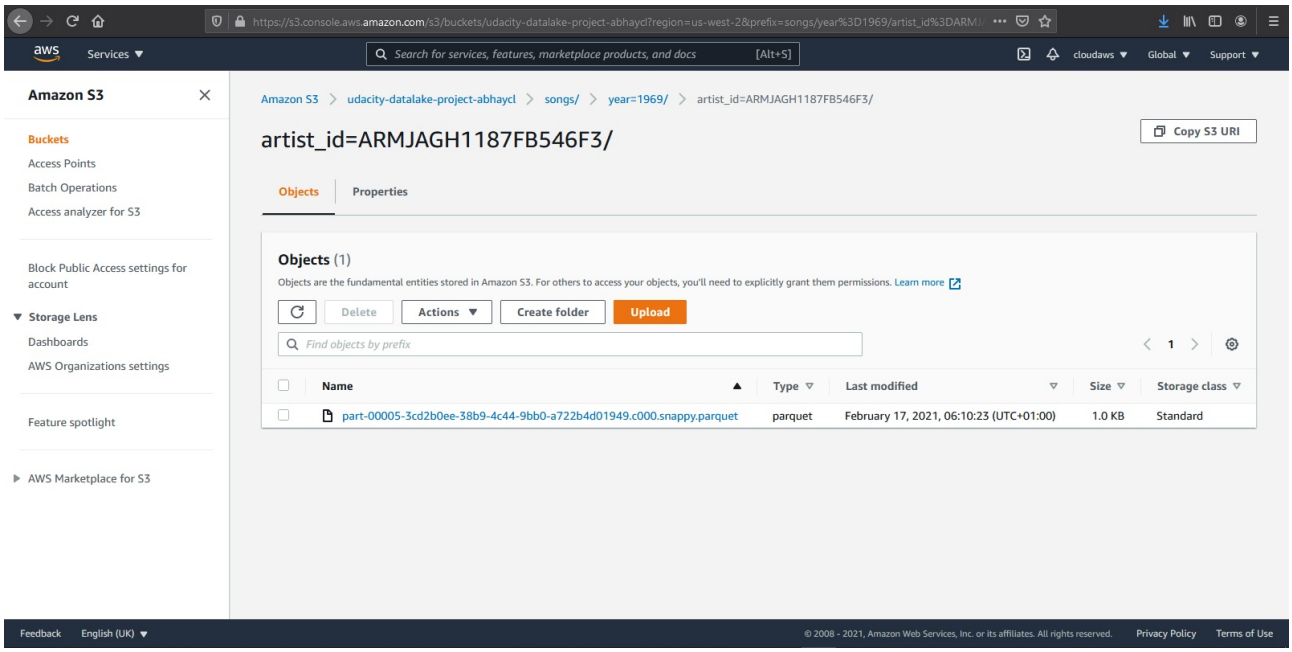
For the songs table, we will extract and transform the song data using only the columns corresponding to the songs table suggested in the project specifications, sorting the records by song ID and without duplicates.

```
songs_table = df.select("song_id", "title", "artist_id", "year", "duration") \
    .orderBy("song_id") \
    .drop_duplicates()
```

```
--- Extraction of columns to create songs table. ---
- Shows the scheme and some songs table records...
root
 |-- song_id: string (nullable = true)
 |-- title: string (nullable = true)
 |-- artist_id: string (nullable = true)
 |-- year: long (nullable = true)
 |-- duration: double (nullable = true)

+------------------+--------------------+------------------+----+---------+
|           song_id|               title|         artist_id|year| duration|
+------------------+--------------------+------------------+----+---------+
|SOABWAP12A8C13F82A|           Take Time|AR5LMPY1187FB573FE|1978|258.89914|
|SOAFBCP12A8C13CC7D|King Of Scurf (20...|ARTC1LV1187B9A4858|1972|301.40036|
|SOAPERH12A58A787DC|The One And Only ...|ARZ5H0P1187B98A1DD|   0|230.42567|
|SOBLFFE12AF72AA5BA|              Scream|ARJNIUY12298900C91|2009| 213.9424|
|SOBRKGM12A8C139EF6|Welcome to the Pl...|ARXQBR11187B98A2CC|1985|821.05424|
+------------------+--------------------+------------------+----+---------+
only showing top 5 rows
```

We proceed to load our data into our destination S3 bucket to write the song table in parquet files partitioned by year and artist.

```
songs_table.write.partitionBy("year", "artist_id").parquet(os.path.join(output_d
```



For the artist table, we will extract and transform the song data using only the columns corresponding to the artist table suggested in the project specifications, sorting the records by artist ID and without duplicates.

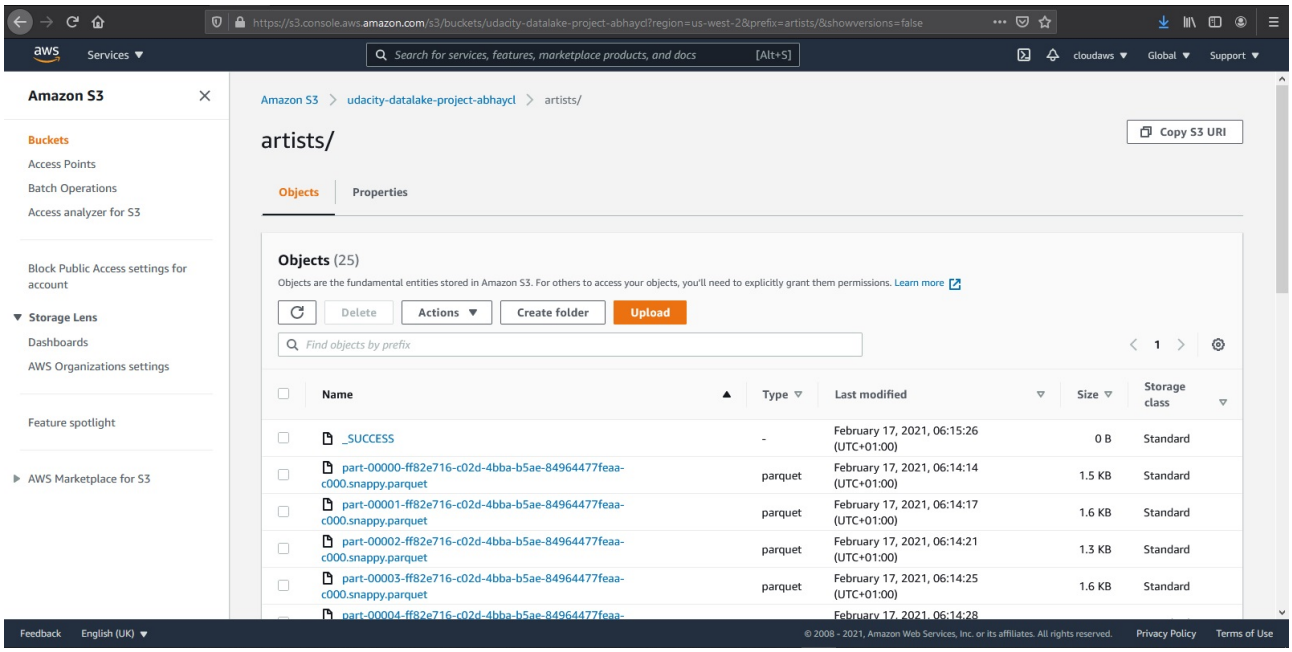
```
artists_table = df.selectExpr("artist_id", "artist_name as name", "artist_locati
    .orderBy("artist_id") \
```

```
        .drop_duplicates()
```

```
--- Extraction of columns to create artists table. ---
- Shows the scheme and some artists table records...
root
 |-- artist_id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- location: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)

+------------------+--------------------+--------------------+--------+----------+
|         artist_id|                name|            location|latitude| longitude|
+------------------+--------------------+--------------------+--------+----------+
|AR0MWD61187B9B2B12|International Noi...|                    |    null|      null|
|AR10USD1187B99F3F1|Tweeterfriendly M...|Burlington, Ontar...|    null|      null|
|AR1C2IX1187B99BF74|     Broken Spindles|                    |    null|      null|
|AR1KTV21187B9ACD72|             Cristina|     California - LA|34.05349|-118.24532|
|AR5LMPY1187FB573FE|   Chaka Khan_ Rufus|         Chicago, IL|41.88415| -87.63241|
+------------------+--------------------+--------------------+--------+----------+
only showing top 5 rows
```

We proceed to load our data into our destination S3 bucket to write the artist table in parquet files.

```
    artists_table.write.parquet(os.path.join(output_data, "artists"), mode="overwrit
```



Process log data (log_data directory).

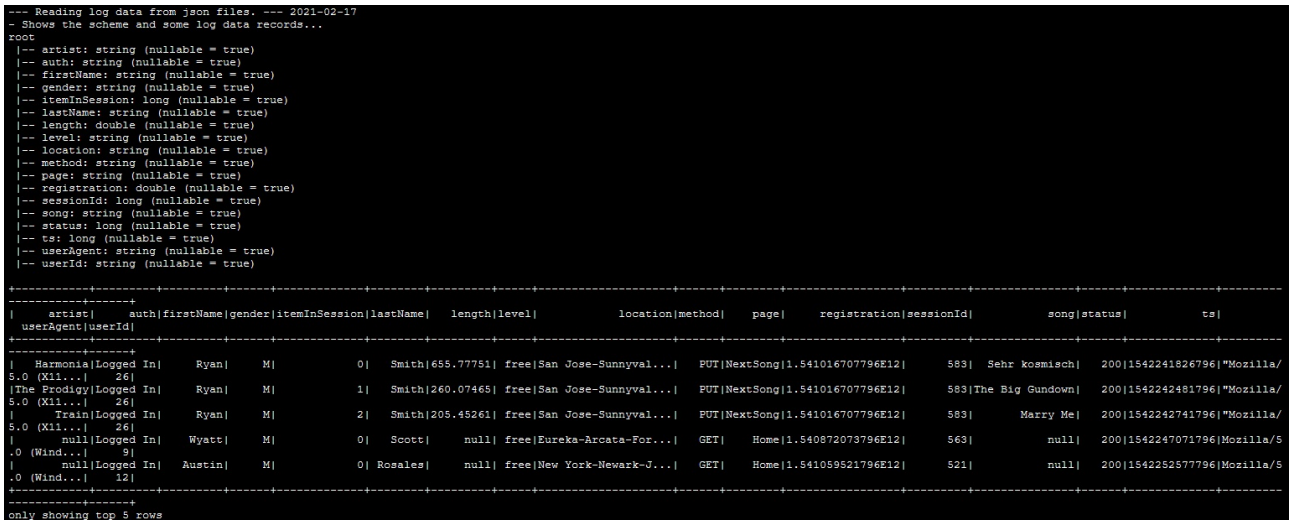
We will perform ETL on the files in log_data directory to create the remaining two dimensional tables: time and users, as well as the songplays fact table.

This is what a single log file looks like:

```
{"artist":null,"auth":"Logged In","firstName":"Walter","gender":"M","itemInSession":
{"artist":null,"auth":"Logged In","firstName":"Kaylee","gender":"F","itemInSession":
{"artist":"Des'ree","auth":"Logged In","firstName":"Kaylee","gender":"F","itemInSess
```

We will proceed to read the data from the source S3 bucket, in the image below we can check the result of the task.

```
df = spark.read.json(log_data)
```



First we filter by actions for song plays.

```
df = df.where("page = 'NextSong'")
```

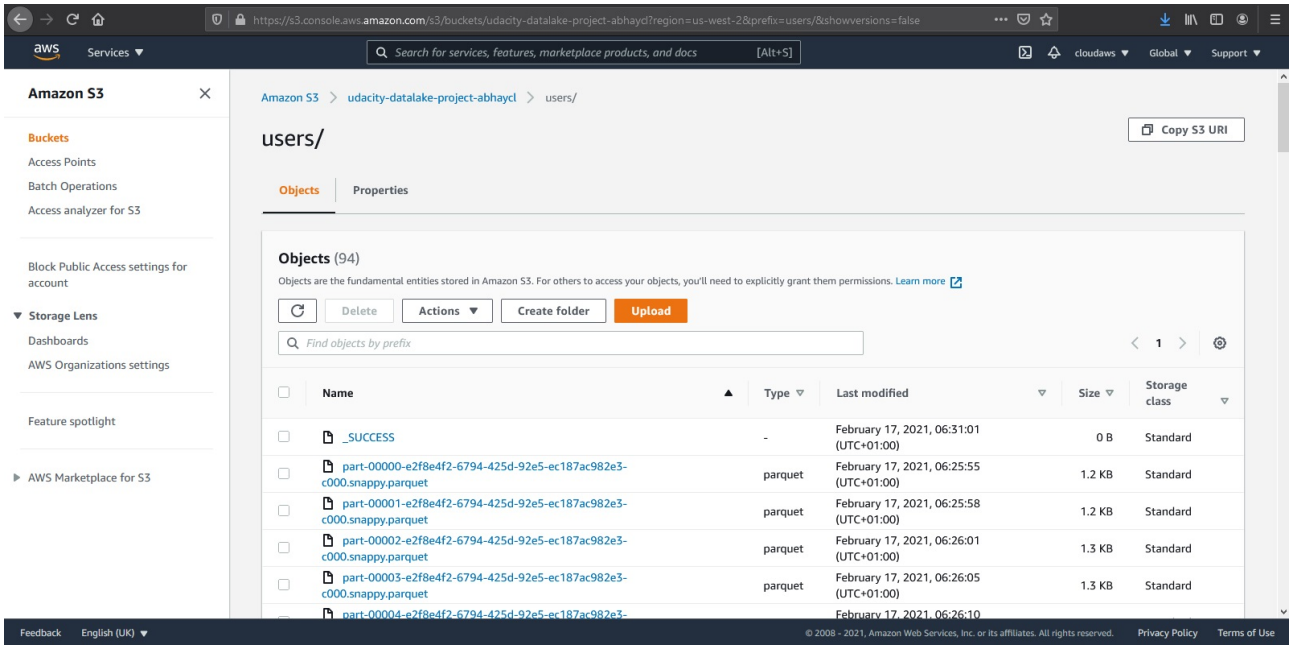
For the users table, we will extract and transform the log data using only the columns corresponding to the users table suggested in the project specifications, filtering the records that have value in the user id, sorting the records by user ID and without duplicates.

```
users_table = df.selectExpr("userId as user_id", "firstName as first_name", "las
    .filter("user_id <> ''") \
    .orderBy("user_id") \
    .drop_duplicates()
```

```
--- Extraction of columns to create users table. ---
- Shows the scheme and some users table records...
root
 |-- user_id: string (nullable = true)
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- level: string (nullable = true)

+-------+----------+---------+------+-----+
|user_id|first_name|last_name|gender|level|
+-------+----------+---------+------+-----+
|     10|    Sylvie|     Cruz|     F| free|
|    100|     Adler|  Barrera|     M| free|
|    101|    Jayden|      Fox|     M| free|
|     11| Christian|   Porter|     F| free|
|     12|    Austin|  Rosales|     M| free|
+-------+----------+---------+------+-----+
only showing top 5 rows
```

We proceed to load our data into our destination S3 bucket to write the users table in parquet files.

```
users_table.write.parquet(os.path.join(output_data, "users"), mode="overwrite")
```



Creation of the timestamp and datetime column from original timestamp column.

```
get_timestamp = udf(lambda x: datetime.fromtimestamp(int(x)/1000).strftime("%Y-%
df = df.withColumn("timestamp", get_timestamp("ts"))

    get_datetime = udf(lambda x: datetime.fromtimestamp(int(x)/1000).strftime("%
df = df.withColumn("datetime", get_datetime("ts"))
```

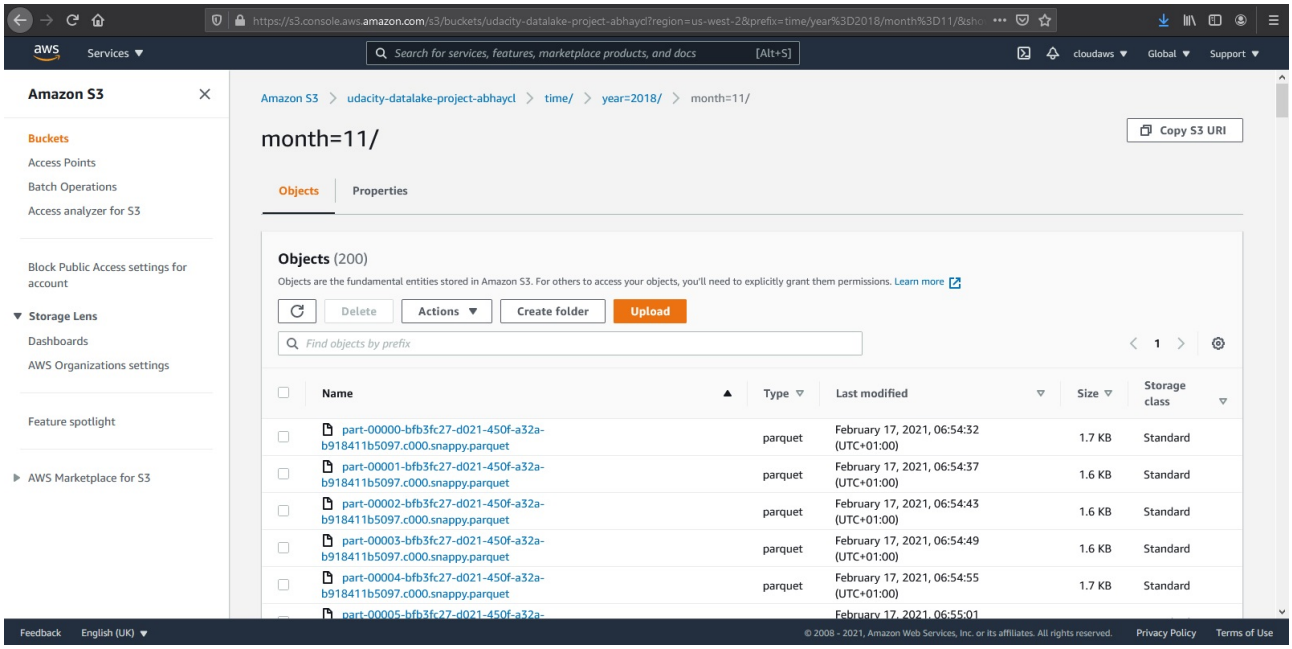
For the time table, we will extract and transform the log data using only the columns corresponding to the time table suggested in the project specifications, sorting the records by start time and without duplicates.

```
time_table = df.select(col("timestamp").alias("start_time"), hour("timestamp").a
    .orderBy("start_time") \
    .drop_duplicates()
```

```
--- Extraction of columns to create time table. ---
- Shows the scheme and some time table records...
root
 |-- start_time: string (nullable = true)
 |-- hour: integer (nullable = true)
 |-- day: integer (nullable = true)
 |-- week: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- year: integer (nullable = true)
 |-- weekday: string (nullable = true)

+-------------------+----+---+----+-----+----+-------+
|         start_time|hour|day|week|month|year|weekday|
+-------------------+----+---+----+-----+----+-------+
|2018-11-01 21:01:46|  21|  1|  44|   11|2018|      4|
|2018-11-01 21:05:52|  21|  1|  44|   11|2018|      4|
|2018-11-01 21:08:16|  21|  1|  44|   11|2018|      4|
|2018-11-01 21:11:13|  21|  1|  44|   11|2018|      4|
|2018-11-01 21:17:33|  21|  1|  44|   11|2018|      4|
+-------------------+----+---+----+-----+----+-------+
only showing top 5 rows
```

We proceed to load our data into our destination S3 bucket to write the song table in parquet files partitioned by year and month.

```
time_table.write.partitionBy("year", "month").parquet(os.path.join(output_da
```



We read, extract and transform the song data to use for songplays table.

```
song_df = spark.read.json(os.path.join(input_data, "song_data/*/*/*/*.json")).se
```

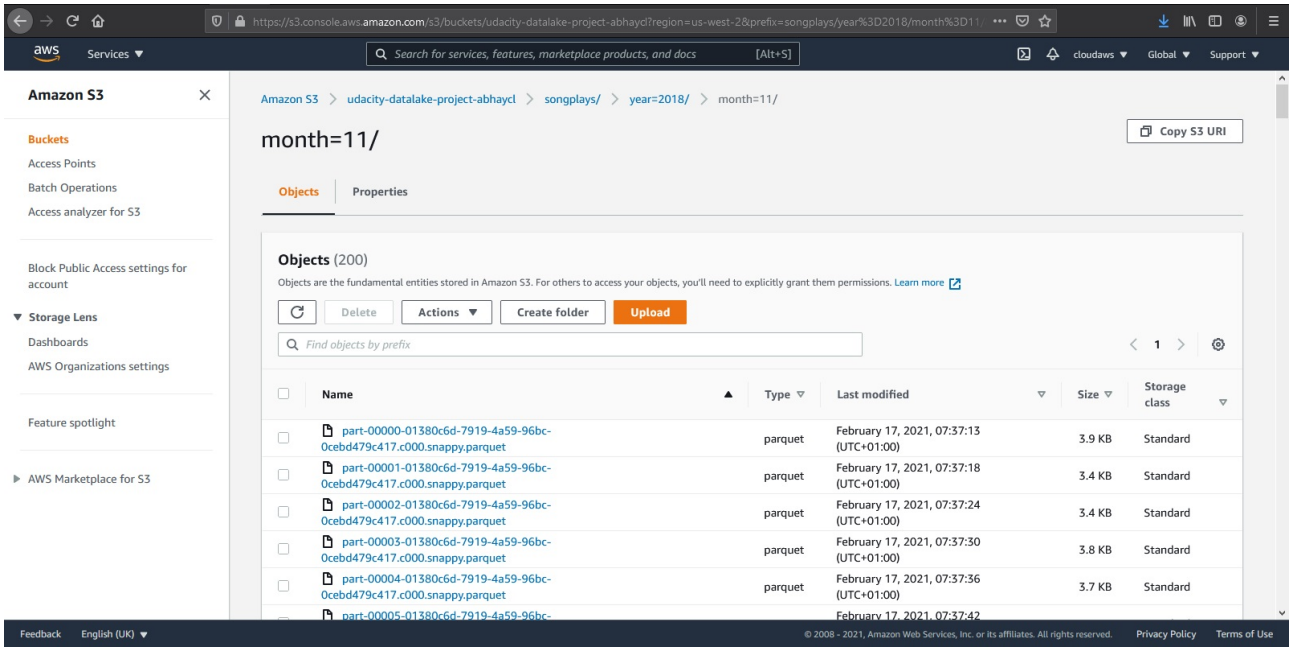
For the songplays table, we will extract and transform from joined song and log datasets using only the columns corresponding to the songplays fact table suggested in the project specifications, sorting the records by start time and user id.

```
songplays_table = df.join(song_df, \
    (df.song == song_df.title) & (df.artist == song_df.artist_name) & (df.length
    .select(df.timestamp.alias("start_time"), df.userId.alias("user_id"), df.lev
    .orderBy("start_time", "user_id") \
    .withColumn("songplay_id", F.monotonically_increasing_id())
```

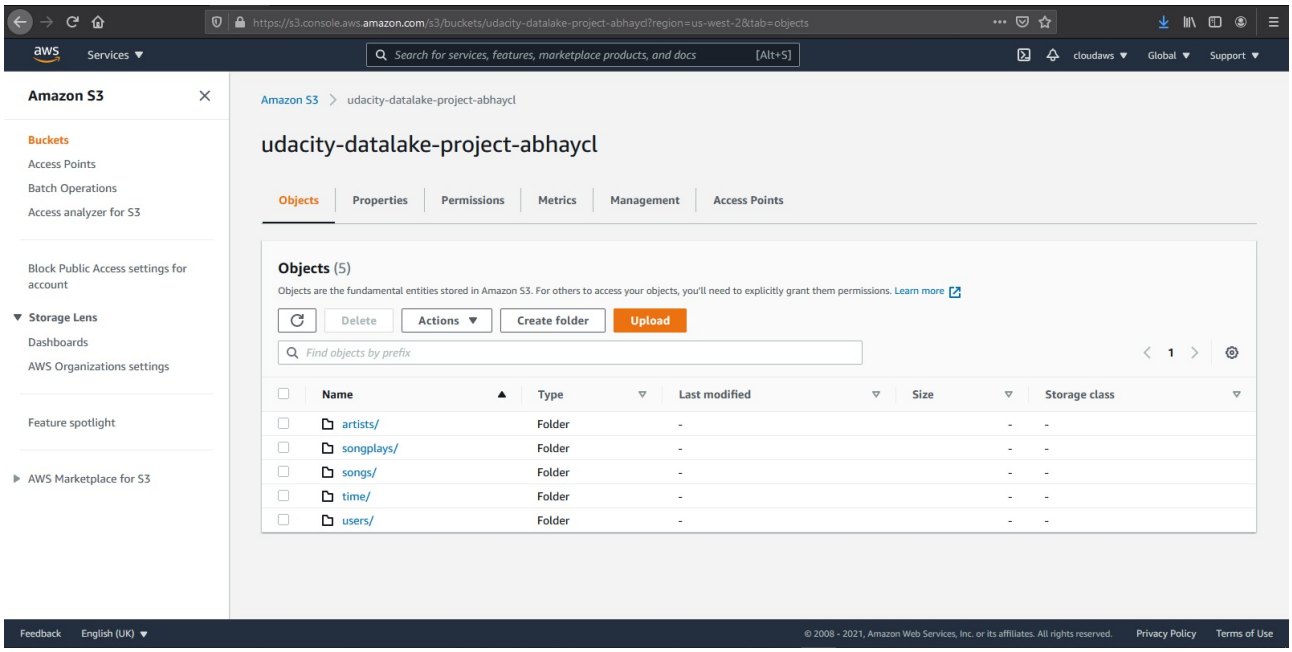
```
--- Extraction of columns to create songplays table. ---
- Shows the scheme and some songplays table records...
root
 |-- start_time: string (nullable = true)
 |-- user_id: string (nullable = true)
 |-- level: string (nullable = true)
 |-- song_id: string (nullable = true)
 |-- artist_id: string (nullable = true)
 |-- session_id: long (nullable = true)
 |-- location: string (nullable = true)
 |-- user_agent: string (nullable = true)
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- songplay_id: long (nullable = false)

+-------------------+-------+-----+-------+---------+----------+--------------------+--------------------+----+-----+-----------+
|         start_time|user_id|level|song_id|artist_id|session_id|            location|          user_agent|year|month|songplay_id|
+-------------------+-------+-----+-------+---------+----------+--------------------+--------------------+----+-----+-----------+
|2018-11-01 21:01:46|      8| free|   null|     null|       139|Phoenix-Mesa-Scot...|"Mozilla/5.0 (Win...|2018|   11|          0|
|2018-11-01 21:05:52|      8| free|   null|     null|       139|Phoenix-Mesa-Scot...|"Mozilla/5.0 (Win...|2018|   11|          1|
|2018-11-01 21:08:16|      8| free|   null|     null|       139|Phoenix-Mesa-Scot...|"Mozilla/5.0 (Win...|2018|   11|          2|
|2018-11-01 21:11:13|      8| free|   null|     null|       139|Phoenix-Mesa-Scot...|"Mozilla/5.0 (Win...|2018|   11|          3|
|2018-11-01 21:17:33|      8| free|   null|     null|       139|Phoenix-Mesa-Scot...|"Mozilla/5.0 (Win...|2018|   11|          4|
+-------------------+-------+-----+-------+---------+----------+--------------------+--------------------+----+-----+-----------+
only showing top 5 rows
```

We proceed to load our data into our destination S3 bucket to write the songplays table in parquet files partitioned by year and month.

```
songplays_table.write.partitionBy("year", "month").parquet(os.path.join(outp
```



In our target bucked S3 we will be able to check all saved data corresponding to the five defined tables.



Conclusion.

As the data engineer of the music streaming startup. We read our data from a source S3 bucked, with our spark cluster we extracted and transformed the data into five tables according to the project specifications and deposited them in our destination S3 bucked for further analysis or data processing.