ArupChakraborty_Assignment2.1

May 20, 2025

1 AWS Data Lake Query Notebook

This notebook sets up the environment, configures Athena, and executes various queries to analyze the ingested data from the AWS S3 Data Lake.

[111]:
```
import boto3
import sagemaker
import pandas as pd

sess = sagemaker.Session()
bucket = sess.default_bucket()
role = sagemaker.get_execution_role()
region = boto3.Session().region_name
account_id = boto3.client("sts").get_caller_identity().get("Account")

sm = boto3.Session().client(service_name="sagemaker", region_name=region)
```

[112]:
```
s3_private_path_tsv = "s3://{}/amazon-reviews-pds/tsv".format(bucket)
print(s3_private_path_tsv)
```

```
s3://sagemaker-us-east-1-672518276407/amazon-reviews-pds/tsv
```

[113]:
```
!aws s3 cp "dataset.csv" $s3_private_path_tsv/
```

```
upload: ./dataset.csv to s3://sagemaker-us-east-1-672518276407/amazon-reviews-
pds/tsv/dataset.csv
```

[114]:
```
print(s3_private_path_tsv)
```

```
s3://sagemaker-us-east-1-672518276407/amazon-reviews-pds/tsv
```

[95]:
```
from pyathena import connect
```

[119]:
```
database_name = "dsoawsv5"
s3_staging_dir = "s3://sagemaker-us-east-1-672518276407/amazon-reviews-pds/tsv"
conn = connect(region_name=region, s3_staging_dir=s3_staging_dir)
statement = "CREATE DATABASE IF NOT EXISTS {}".format(database_name)
print(statement)
```

```
CREATE DATABASE IF NOT EXISTS dsoawsV4
```

[120]:
```
import pandas as pd

pd.read_sql(statement, conn)
```

```
/tmp/ipykernel_3285/3803073958.py:3: UserWarning: pandas only supports
SQLAlchemy connectable (engine/connection) or database string URI or sqlite3
DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using
SQLAlchemy.
  pd.read_sql(statement, conn)
```

[120]:
```
Empty DataFrame
Columns: []
Index: []
```

[121]:
```
statement = "SHOW DATABASES"

df_show = pd.read_sql(statement, conn)
df_show.head(10)
```

```
/tmp/ipykernel_3285/3999478089.py:3: UserWarning: pandas only supports
SQLAlchemy connectable (engine/connection) or database string URI or sqlite3
DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using
SQLAlchemy.
  df_show = pd.read_sql(statement, conn)
```

[121]:

	database_name
0	default
1	dsoaws
2	dsoawsv2
3	dsoawsv3
4	dsoawsv4

1.1 Fix the issue with dataset.csv file

[135]:
```
import pandas as pd

# Path to the uploaded CSV file
csv_path = "./dataset.csv"

try:
    # Attempt to read the CSV file with common delimiters and encodings
    df_comma = pd.read_csv(csv_path)
    df_tab = pd.read_csv(csv_path, delimiter='\t')
    df_pipe = pd.read_csv(csv_path, delimiter='|')

    # Display the first few rows and column names for each delimiter
```

```
    print("Comma Delimiter:")
    print("Columns:", df_comma.columns.tolist())
    print(df_comma.head(), "\n")

    print("Tab Delimiter:")
    print("Columns:", df_tab.columns.tolist())
    print(df_tab.head(), "\n")

    print("Pipe Delimiter:")
    print("Columns:", df_pipe.columns.tolist())
    print(df_pipe.head(), "\n")

    # Check encoding issues
    with open(csv_path, 'rb') as file:
        raw_data = file.read(1000)  # Read a sample of the file
        print("Sample Encoding Inspection:")
        print(raw_data[:200])  # Print the first 200 bytes
except Exception as e:
    print(f"Error occurred while inspecting CSV: {e}")
```

```
Error occurred while inspecting CSV: Error tokenizing data. C error: Expected 1
fields in line 715, saw 2
```

[136]:
```
import pandas as pd

# Path to the uploaded CSV file
csv_path = "./dataset.csv"

try:
    # Open the file and inspect the problematic lines manually
    with open(csv_path, 'r', encoding='utf-8') as file:
        lines = file.readlines()

    # Display lines around the problematic line (715)
    print("Lines around the problematic area:")
    for i in range(710, 720):
        print(f"Line {i+1}: {lines[i].strip()}")
except Exception as e:
    print(f"Error occurred while reading CSV lines: {e}")
```

```
Lines around the problematic area:
Line 711: 709,1qHiDbTJI7GB62W3BBFigx,A Great Big World;Futuristic,When the
Morning Comes,Hold Each Other (feat. Futuristic),47,216040,False,0.61,0.796,10,-
5.431,1,0.0656,0.0365,0.0,0.274,0.51,159.944,4,acoustic
Line 712: 710,5msvkAkQ8o2GhNxOu3YW5D,Hanare Gumi,  ,   ,28,274133,False,0.63,
0.367,1,-11.054,1,0.0232,0.052,0.000202,0.103,0.547,98.995,4,acoustic
Line 713: 711,3YllcvA3PW1DUwjckCVjIw,Ray LaMontagne,MONOVISION,I Was Born To
```

```
Love You,60,251933,False,0.659,0.236,9,-12.56,1,0.0304,0.84,6.32e-
05,0.206,0.216,78.088,4,acoustic
Line 714: 712,2cTaSKEc8OZdF6Tpg2QQsS,Aimyon,Heard that there's good pasta,Naked
Heart,47,296653,False,0.559,0.448,8,-4.11,1,0.0275,0.537,1.85e-
05,0.0915,0.354,148.02,4,acoustic
Line 715: 713,1z2fSrYZqrO5tMqzULn9OD,Tyrone Wells,"The ""Hits"" | Acoustic",Days
I Will Remember,30,201506,False,0.804,0.457,6,-
7.845,1,0.197,0.6,0.0,0.15,0.552,98.035,4,acoustic
Line 716: 714,5QRt4QovcONoIVzwnlkU7z,Kaiak,River Love,Smell the Coffee,49,201861
,False,0.637,0.46,3,-9.07,1,0.0384,0.855,4.8e-05,0.114,0.345,93.955,4,acoustic
Line 717: 715,7Igc6JBOxPDcn5yRkkVGQX,Jason Mraz,Know.,Unlonely,51,231266,False,0
.795,0.667,5,-4.831,0,0.0392,0.379,0.0,0.221,0.754,97.987,4,acoustic
Line 718: 716,3XfMyT4Xf5LegDhvbrFEjp,Boyce Avenue;Bea Miller,"Cover Sessions,
Vol. 4",See You Again,50,239802,False,0.57,0.413,9,-
7.034,1,0.0345,0.535,0.0,0.076,0.278,152.032,4,acoustic
Line 719: 717,3IvEnBPF1egP9y1q3C2Fg4,Sakura Fujiwara,SUPERMARKET,   ,28,247666
,False,0.565,0.566,6,-
7.258,1,0.0272,0.472,0.000426,0.148,0.551,92.017,4,acoustic
Line 720: 718,4XBkxG1jC5RBQJt65gma8v,Filip Nordin,Acustic Covers of Popular
Songs,More Than You Know - Acoustic Covers of Popular Songs,51,155510,False,0.77
6,0.319,8,-7.496,1,0.031,0.791,0.0,0.159,0.501,98.016,4,acoustic
```

[140]:
```
import pandas as pd

# Path to the uploaded CSV file
csv_path = "./dataset.csv"

try:
    # Correctly load the CSV file by handling embedded quotes and commas
    df = pd.read_csv(csv_path, quotechar='"', escapechar='\\', delimiter=',')
    print("CSV loaded successfully with proper quote handling.")
    print(df.head(10))
except Exception as e:
    print(f"Error occurred while loading CSV: {e}")
```

```
CSV loaded successfully with proper quote handling.
   Unnamed: 0                track_id                                artists  \
0           0  5SuOikwiRyPMVoIQDJUgSV                            Gen Hoshino
1           1  4qPNDBW1i3p13qLCt0Ki3A                           Ben Woodward
2           2  1iJBSr7s7jYXzM8EGcbK5b                 Ingrid Michaelson;ZAYN
3           3  6lfxq3CG4xtTiEg7opyCyx                            Kina Grannis
4           4  5vjLSffimiIP26QG5WcN2K                       Chord Overstreet
5           5  01MVOl9KtVTNfFiBU9I7dc                           Tyrone Wells
6           6  6Vc5wAMmXdKIAM7WUoEb7N  A Great Big World;Christina Aguilera
7           7  1EzrEOXmMH3G43AXT1y7pA                             Jason Mraz
8           8  0IktbUcnAGrvD03AWnz3Q8             Jason Mraz;Colbie Caillat
9           9  7k9GuJYLp2AzqokyEdwEw2                          Ross Copperman
```

	album_name	\
0	Comedy	
1	Ghost (Acoustic)	
2	To Begin Again	
3	Crazy Rich Asians (Original Motion Picture Sou…	
4	Hold On	
5	Days I Will Remember	
6	Is There Anybody Out There?	
7	We Sing. We Dance. We Steal Things.	
8	We Sing. We Dance. We Steal Things.	
9	Hunger	

	track_name	popularity	duration_ms	explicit	\
0	Comedy	73	230666	False	
1	Ghost - Acoustic	55	149610	False	
2	To Begin Again	57	210826	False	
3	Can't Help Falling In Love	71	201933	False	
4	Hold On	82	198853	False	
5	Days I Will Remember	58	214240	False	
6	Say Something	74	229400	False	
7	I'm Yours	80	242946	False	
8	Lucky	74	189613	False	
9	Hunger	56	205594	False	

	danceability	energy	…	loudness	mode	speechiness	acousticness	\
0	0.676	0.4610	…	-6.746	0	0.1430	0.0322	
1	0.420	0.1660	…	-17.235	1	0.0763	0.9240	
2	0.438	0.3590	…	-9.734	1	0.0557	0.2100	
3	0.266	0.0596	…	-18.515	1	0.0363	0.9050	
4	0.618	0.4430	…	-9.681	1	0.0526	0.4690	
5	0.688	0.4810	…	-8.807	1	0.1050	0.2890	
6	0.407	0.1470	…	-8.822	1	0.0355	0.8570	
7	0.703	0.4440	…	-9.331	1	0.0417	0.5590	
8	0.625	0.4140	…	-8.700	1	0.0369	0.2940	
9	0.442	0.6320	…	-6.770	1	0.0295	0.4260	

	instrumentalness	liveness	valence	tempo	time_signature	track_genre
0	0.000001	0.3580	0.7150	87.917	4	acoustic
1	0.000006	0.1010	0.2670	77.489	4	acoustic
2	0.000000	0.1170	0.1200	76.332	4	acoustic
3	0.000071	0.1320	0.1430	181.740	3	acoustic
4	0.000000	0.0829	0.1670	119.949	4	acoustic
5	0.000000	0.1890	0.6660	98.017	4	acoustic
6	0.000003	0.0913	0.0765	141.284	3	acoustic
7	0.000000	0.0973	0.7120	150.960	4	acoustic
8	0.000000	0.1510	0.6690	130.088	4	acoustic
9	0.004190	0.0735	0.1960	78.899	4	acoustic

```
[10 rows x 21 columns]
```

1.2 Use the cleaned csv file for creating datalake on AWS Athena

[142]:
```
csv_output_path = "./cleaned_dataset.csv"
df.to_csv(csv_output_path, index=False, quotechar='"', escapechar='\\')
print(f"DataFrame saved to {csv_output_path}")
```

```
DataFrame saved to ./cleaned_dataset.csv
```

[147]:
```
s3_staging_dir = "s3://sagemaker-us-east-1-672518276407/amazon-reviews-pds-1/
 ↪tsv"
!aws s3 cp "cleaned_dataset.csv" $s3_staging_dir/
```

```
upload: ./cleaned_dataset.csv to s3://sagemaker-us-east-1-672518276407/amazon-
reviews-pds-1/tsv/cleaned_dataset.csv
```

[144]:
```
print(s3_private_path_tsv)
```

```
s3://sagemaker-us-east-1-672518276407/amazon-reviews-pds/tsv
```

[158]:
```
database_name = "dsoaws_new"
conn = connect(region_name=region, s3_staging_dir=s3_staging_dir)
statement = "CREATE DATABASE IF NOT EXISTS {}".format(database_name)
print(statement)
pd.read_sql(statement, conn)
```

```
CREATE DATABASE IF NOT EXISTS dsoaws_new
```

```
/tmp/ipykernel_3285/750785309.py:5: UserWarning: pandas only supports SQLAlchemy
connectable (engine/connection) or database string URI or sqlite3 DBAPI2
connection. Other DBAPI2 objects are not tested. Please consider using
SQLAlchemy.
  pd.read_sql(statement, conn)
```

[158]:
```
Empty DataFrame
Columns: []
Index: []
```

[159]:
```
statement = "SHOW DATABASES"
df_show = pd.read_sql(statement, conn)
df_show.head(20)
```

```
/tmp/ipykernel_3285/1371328689.py:2: UserWarning: pandas only supports
SQLAlchemy connectable (engine/connection) or database string URI or sqlite3
DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using
SQLAlchemy.
  df_show = pd.read_sql(statement, conn)
```

[159]:
```
   database_name
0        default
1         dsoaws
2      dsoaws001
3        dsoaws2
4        dsoaws3
5        dsoaws4
6     dsoaws_001
7     dsoaws_new
8       dsoawsv2
9       dsoawsv3
10      dsoawsv4
11      dsoawsv6
12      dsoawsv8
```

[73]:
```
import boto3
import pandas as pd
from io import StringIO

# S3 bucket and file path
bucket_name = "sagemaker-us-east-1-672518276407"
file_key = "amazon-reviews-pds/tsv/dataset.csv"

# Create an S3 client
s3 = boto3.client("s3")

try:
    # Fetch the CSV file from S3
    response = s3.get_object(Bucket=bucket_name, Key=file_key)

    # Read the content of the file
    content = response["Body"].read().decode("utf-8")

    # Load the content into a DataFrame
    data = pd.read_csv(StringIO(content))

    # Display the first few rows
    print(data.head())
except Exception as e:
    print(f"Error occurred: {e}")
```

```
   Unnamed: 0                track_id                artists  \
0           0  5SuOikwiRyPMVoIQDJUgSV             Gen Hoshino
1           1  4qPNDBW1i3p13qLCt0Ki3A            Ben Woodward
2           2  1iJBSr7s7jYXzM8EGcbK5b  Ingrid Michaelson;ZAYN
3           3  6lfxq3CG4xtTiEg7opyCyx            Kina Grannis
4           4  5vjLSffimiIP26QG5WcN2K        Chord Overstreet
```

```
                                        album_name  \
0                                           Comedy
1                                  Ghost (Acoustic)
2                                    To Begin Again
3  Crazy Rich Asians (Original Motion Picture Sou…
4                                        Hold On

                  track_name  popularity  duration_ms  explicit  \
0                     Comedy          73       230666     False
1          Ghost - Acoustic           55       149610     False
2            To Begin Again           57       210826     False
3  Can't Help Falling In Love          71       201933     False
4                    Hold On           82       198853     False

   danceability  energy  …  loudness  mode  speechiness  acousticness  \
0         0.676  0.4610  …    -6.746     0       0.1430        0.0322
1         0.420  0.1660  …   -17.235     1       0.0763        0.9240
2         0.438  0.3590  …    -9.734     1       0.0557        0.2100
3         0.266  0.0596  …   -18.515     1       0.0363        0.9050
4         0.618  0.4430  …    -9.681     1       0.0526        0.4690

   instrumentalness  liveness  valence    tempo  time_signature  track_genre
0          0.000001    0.3580    0.715   87.917               4     acoustic
1          0.000006    0.1010    0.267   77.489               4     acoustic
2          0.000000    0.1170    0.120   76.332               4     acoustic
3          0.000071    0.1320    0.143  181.740               3     acoustic
4          0.000000    0.0829    0.167  119.949               4     acoustic

[5 rows x 21 columns]
```

[74]:
```
# Filter songs with popularity greater than or equal to 99
df_high_popularity = data[data['popularity'] >= 99]
df_high_popularity = df_high_popularity[['artists', 'track_name', 'popularity']]
print("Songs with popularity  99:")
print(df_high_popularity)
```

```
Songs with popularity  99:
                  artists                          track_name  popularity
20001  Sam Smith;Kim Petras            Unholy (feat. Kim Petras)         100
51664      Bizarrap;Quevedo  Quevedo: Bzrp Music Sessions, Vol. 52          99
81051  Sam Smith;Kim Petras            Unholy (feat. Kim Petras)         100
```

[75]:
```
# Group by artist and calculate the average popularity
df_avg_popularity = data.groupby('artists')['popularity'].mean().reset_index()

# Filter artists with an average popularity of 92
```

```
df_avg_popularity_92 = df_avg_popularity[df_avg_popularity['popularity'] == 92]
print("Artists with an average popularity of 92:")
print(df_avg_popularity_92)
```

```
Artists with an average popularity of 92:
                artists  popularity
11491       Harry Styles        92.0
22845  Rema;Selena Gomez        92.0
```

[76]:
```
# Group by genre and calculate the average energy
df_avg_energy = data.groupby('track_genre')['energy'].mean().reset_index()

# Get the top 10 genres by average energy
df_top10_genres = df_avg_energy.sort_values(by='energy', ascending=False).
 ↪head(10)
print("Top 10 genres with the highest average energy:")
print(df_top10_genres)
```

```
Top 10 genres with the highest average energy:
      track_genre    energy
22    death-metal  0.931470
42      grindcore  0.924201
72      metalcore  0.914485
46          happy  0.910971
49      hardstyle  0.901246
27  drum-and-bass  0.876635
6     black-metal  0.874897
50    heavy-metal  0.874003
78          party  0.871237
61         j-idol  0.868677
```

[192]:
```
# Filter tracks by the artist 'Bad Bunny'
bad_bunny_count = data[data['artists'].str.contains('Bad Bunny', na=False)].
 ↪shape[0]
print(f"Number of tracks featuring Bad Bunny: {bad_bunny_count}")
```

```
Number of tracks featuring Bad Bunny: 416
```

[78]:
```
# Group by genre and find the maximum popularity within each genre
df_genre_popularity = data.groupby('track_genre')['popularity'].max().
 ↪reset_index()

# Sort by popularity and get the top 10 genres
df_top10_popular_genres = df_genre_popularity.sort_values(by='popularity',␣
 ↪ascending=False).head(10)
print("Top 10 genres by their most popular track:")
print(df_top10_popular_genres)
```

```
Top 10 genres by their most popular track:
   track_genre  popularity
80         pop         100
20       dance         100
51     hip-hop          99
68      latino          98
89   reggaeton          98
30         edm          98
67       latin          98
88      reggae          98
79       piano          96
90        rock          96
```

[79]:
```
statement = "SHOW TABLES in {}".format('dsoaws')

df_show = pd.read_sql(statement, conn)
df_show.head(5)
```

```
/tmp/ipykernel_3285/2205842015.py:3: UserWarning: pandas only supports
SQLAlchemy connectable (engine/connection) or database string URI or sqlite3
DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using
SQLAlchemy.
  df_show = pd.read_sql(statement, conn)
```

[79]:

	tab_name
0	amazon_reviews_tsv

[80]:
```
# SageMaker session setup
sess = sagemaker.Session()
bucket = sess.default_bucket()
role = sagemaker.get_execution_role()
region = boto3.Session().region_name

# Athena connection setup
database_name = "dsoaws"
table_name_tsv = "amazon_reviews_tsv"

conn = connect(region_name=region, s3_staging_dir=s3_staging_dir)
print("Setup completed.")
```

```
Setup completed.
```

[81]:
```
# Athena connection setup
conn = connect(region_name=region, s3_staging_dir=s3_staging_dir)

# Improved execute_query function
def execute_query(query, description="Query Result"):
    """
```

```
    Executes a Pandas query or calculation and prints the result.

    Parameters:
        query (pd.DataFrame or int): The result of the query.
        description (str): A brief description of the query result.
    """
    try:
        print(f"\n{description}")
        print("=" * len(description))

        # Check if the query returned a DataFrame
        if isinstance(query, pd.DataFrame):
            if not query.empty:
                display(query.head(100))  # Display the first 100 rows
                print(f"Total Records: {len(query)}")
            else:
                print("No matching records found.")
        # Check if the query returned a single value (like a count)
        elif isinstance(query, int):
            print(f"Result: {query}")
        else:
            print("Unexpected query result type.")
    except Exception as e:
        print(f"Error executing query: {e}")
```

1.3 List artist, track_name, and popularity for songs with popularity 99

[82]:
```
import pandasql as psql

# SQL-like query on the DataFrame
query = """
SELECT artists, track_name, popularity
FROM data
WHERE popularity >= 99
"""
result = psql.sqldf(query, locals())
execute_query(result, "Songs with Popularity  99 (SQL-like)")
```

```
Songs with Popularity  99 (SQL-like)
====================================
```

	artists	track_name	popularity
0	Sam Smith;Kim Petras	Unholy (feat. Kim Petras)	100
1	Bizarrap;Quevedo	Quevedo: Bzrp Music Sessions, Vol. 52	99
2	Sam Smith;Kim Petras	Unholy (feat. Kim Petras)	100

```
Total Records: 3
```

1.4 List artists with an average popularity of 92

[83]:
```
query = """
SELECT artists, AVG(popularity) AS avg_popularity
FROM data
GROUP BY artists
HAVING AVG(popularity) = 92
"""
result = psql.sqldf(query, locals())
execute_query(result, "List artists with an average popularity of 92")
```

```
List artists with an average popularity of 92
==============================================

             artists  avg_popularity
0       Harry Styles            92.0
1  Rema;Selena Gomez            92.0

Total Records: 2
```

1.5 List the Top 10 genres with the highest average energy

[84]:
```
query = """
SELECT track_genre, AVG(energy) AS avg_energy
FROM data
GROUP BY track_genre
ORDER BY avg_energy DESC
LIMIT 10
"""
result = psql.sqldf(query, locals())
execute_query(result, "List the Top 10 genres with the highest average energy")
```

```
List the Top 10 genres with the highest average energy
======================================================

     track_genre  avg_energy
0    death-metal    0.931470
1      grindcore    0.924201
2      metalcore    0.914485
3          happy    0.910971
4      hardstyle    0.901246
5  drum-and-bass    0.876635
6    black-metal    0.874897
7    heavy-metal    0.874003
8          party    0.871237
9         j-idol    0.868677

Total Records: 10
```

1.6 How many tracks is Bad Bunny on?

[193]:
```
query = """
SELECT COUNT(*)
FROM data
WHERE artists like '%Bad Bunny%'
"""
result = psql.sqldf(query, locals())
execute_query(result, "How many tracks is Bad Bunny on?")
```

```
How many tracks is Bad Bunny on?
================================

   COUNT(*)
0       416

Total Records: 1
```

1.7 Show the top 10 genres in terms of popularity, sorted by their most popular track

[86]:
```
query = """
SELECT track_genre, MAX(popularity) AS max_popularity
FROM data
GROUP BY track_genre
ORDER BY max_popularity DESC
LIMIT 10
"""
result = psql.sqldf(query, locals())
execute_query(result, "Show the top 10 genres in terms of popularity, sorted by␣
 ↪their most popular track")
```

```
Show the top 10 genres in terms of popularity, sorted by their most popular
track
================================================================================
=

  track_genre  max_popularity
0         pop             100
1       dance             100
2     hip-hop              99
3   reggaeton              98
4      reggae              98
5      latino              98
6       latin              98
7         edm              98
8        rock              96
```

```
9       piano               96

Total Records: 10
```

[87]:
```
sess = sagemaker.Session()
bucket = sess.default_bucket()
role = sagemaker.get_execution_role()
region = boto3.Session().region_name

sm = boto3.Session().client(service_name="sagemaker", region_name=region)
```

[178]:
```
import awswrangler as wr

# Database name
database_name = "dsoaws_parquet"

# Create the Glue database if it doesn't exist
try:
    wr.catalog.create_database(name=database_name)
    print(f"Database '{database_name}' created successfully or already exists.")
except Exception as e:
    print(f"Error creating database: {e}")
```

```
Database 'dsoaws_parquet' created successfully or already exists.
```

[179]:
```
import pandas as pd
import awswrangler as wr

# Load the DataFrame from the CSV file
csv_path = "./cleaned_dataset.csv"
df = pd.read_csv(csv_path)

# S3 bucket and file path for Parquet
bucket_name = "sagemaker-us-east-1-672518276407"
parquet_path = f"s3://{bucket_name}/amazon-reviews-pds-1/parquet/
 ↪cleaned_dataset.parquet"

# Save DataFrame as Parquet to S3
try:
    wr.s3.to_parquet(
        df=df,
        path=parquet_path,
        dataset=True,
        mode="overwrite",
        database="dsoaws_parquet",
        table="amazon_reviews_parquet"
    )
    print(f"Data uploaded successfully to {parquet_path} as Parquet.")
```

```
except Exception as e:
    print(f"Error uploading DataFrame to S3: {e}")
```

```
Data uploaded successfully to s3://sagemaker-us-east-1-672518276407/amazon-
reviews-pds-1/parquet/cleaned_dataset.parquet as Parquet.
```

[182]:
```
# List all tables in the specified database
import awswrangler as wr

# Database name
database_name = "dsoaws_parquet"

# List all tables in the specified database
try:
    tables = list(wr.catalog.get_tables(database=database_name))
    print("Tables in the database:")
    for table in tables:
        print(f"- {table['Name']}")
except Exception as e:
    print(f"Error retrieving tables: {e}")
```

```
Tables in the database:
- amazon_reviews_parquet
```

[183]:
```
# Query the table to check if the data is loaded
try:
    query = f"SELECT * FROM {database_name}.amazon_reviews_parquet LIMIT 5"
    df_sample = wr.athena.read_sql_query(query, database=database_name)
    print("Sample data from the Athena table:")
    print(df_sample)
except Exception as e:
    print(f"Error querying table: {e}")
```

```
Sample data from the Athena table:
   unnamed_0                track_id                 artists  \
0          0  5SuOikwiRyPMVoIQDJUgSV             Gen Hoshino
1          1  4qPNDBW1i3p13qLCt0Ki3A            Ben Woodward
2          2  1iJBSr7s7jYXzM8EGcbK5b  Ingrid Michaelson;ZAYN
3          3  6lfxq3CG4xtTiEg7opyCyx            Kina Grannis
4          4  5vjLSffimiIP26QG5WcN2K        Chord Overstreet

                                    album_name  \
0                                       Comedy
1                            Ghost (Acoustic)
2                              To Begin Again
3  Crazy Rich Asians (Original Motion Picture Sou…
4                                      Hold On
```

```
                  track_name  popularity  duration_ms  explicit  \
0                     Comedy          73       230666     False
1          Ghost - Acoustic           55       149610     False
2            To Begin Again           57       210826     False
3  Can't Help Falling In Love           71       201933     False
4                    Hold On           82       198853     False

   danceability  energy  …  loudness  mode  speechiness  acousticness  \
0         0.676  0.4610  …    -6.746     0       0.1430        0.0322
1         0.420  0.1660  …   -17.235     1       0.0763        0.9240
2         0.438  0.3590  …    -9.734     1       0.0557        0.2100
3         0.266  0.0596  …   -18.515     1       0.0363        0.9050
4         0.618  0.4430  …    -9.681     1       0.0526        0.4690

   instrumentalness  liveness  valence    tempo  time_signature  track_genre
0          0.000001    0.3580    0.715   87.917               4     acoustic
1          0.000006    0.1010    0.267   77.489               4     acoustic
2          0.000000    0.1170    0.120   76.332               4     acoustic
3          0.000071    0.1320    0.143  181.740               3     acoustic
4          0.000000    0.0829    0.167  119.949               4     acoustic

[5 rows x 21 columns]
```

1.8 List artist, track_name, and popularity for songs that have a popularity greater than or equal to 99

[185]:
```
query1 = """
SELECT artists, track_name, popularity
FROM dsoaws_parquet.amazon_reviews_parquet
WHERE popularity >= 99
"""
df1 = wr.athena.read_sql_query(query1, database="dsoaws_parquet")
print("Songs with popularity >= 99:")
print(df1)
```

```
Songs with popularity >= 99:
               artists                                track_name  popularity
0  Sam Smith;Kim Petras              Unholy (feat. Kim Petras)         100
1     Bizarrap;Quevedo  Quevedo: Bzrp Music Sessions, Vol. 52          99
2  Sam Smith;Kim Petras              Unholy (feat. Kim Petras)         100
```

1.9 List artists with an average popularity of 92

[186]:
```
query2 = """
SELECT artists, AVG(popularity) AS avg_popularity
FROM dsoaws_parquet.amazon_reviews_parquet
GROUP BY artists
```

```
HAVING AVG(popularity) = 92
"""
df2 = wr.athena.read_sql_query(query2, database="dsoaws_parquet")
print("Artists with an average popularity of 92:")
print(df2)
```

```
Artists with an average popularity of 92:
             artists  avg_popularity
0  Rema;Selena Gomez            92.0
1       Harry Styles            92.0
```

1.10 List the Top 10 genres with the highest average energy

[187]:
```
query3 = """
SELECT track_genre, AVG(energy) AS avg_energy
FROM dsoaws_parquet.amazon_reviews_parquet
GROUP BY track_genre
ORDER BY avg_energy DESC
LIMIT 10
"""
df3 = wr.athena.read_sql_query(query3, database="dsoaws_parquet")
print("Top 10 genres with highest average energy:")
print(df3)
```

```
Top 10 genres with highest average energy:
     track_genre  avg_energy
0    death-metal    0.931470
1      grindcore    0.924201
2      metalcore    0.914485
3          happy    0.910971
4      hardstyle    0.901246
5  drum-and-bass    0.876635
6    black-metal    0.874897
7    heavy-metal    0.874003
8          party    0.871237
9         j-idol    0.868677
```

1.11 How many tracks is Bad Bunny on?

[194]:
```
query4 = """
SELECT COUNT(*) AS track_count
FROM dsoaws_parquet.amazon_reviews_parquet
WHERE artists LIKE '%Bad Bunny%'
"""
df4 = wr.athena.read_sql_query(query4, database="dsoaws_parquet")
print("Number of tracks featuring Bad Bunny:")
print(df4)
```

```
Number of tracks featuring Bad Bunny:
   track_count
0          416
```

1.12 Show the top 10 genres in terms of popularity, sorted by their most popular track

[189]:
```
query5 = """
SELECT track_genre, MAX(popularity) AS max_popularity
FROM dsoaws_parquet.amazon_reviews_parquet
GROUP BY track_genre
ORDER BY max_popularity DESC
LIMIT 10
"""
df5 = wr.athena.read_sql_query(query5, database="dsoaws_parquet")
print("Top 10 genres by most popular track:")
print(df5)
```

```
Top 10 genres by most popular track:
  track_genre  max_popularity
0         pop             100
1       dance             100
2     hip-hop              99
3      reggae              98
4   reggaeton              98
5         edm              98
6       latin              98
7      latino              98
8       piano              96
9        rock              96
```

[ ]:
```

```