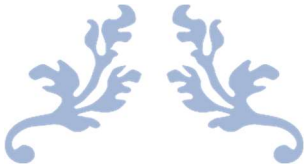


STREAMING SERIALIZATION

USING SPARK

THIRUVENI B
2320446

Streaming Data Serialization: Serialize streaming data into various formats (e.g., JSON) for storage or exchange

METHOD: Streaming Data Serialization

ABSTRACT

In the realm of data manipulation and analysis using Apache Spark's DataFrame API, efficient access to metadata and manipulation of DataFrame attributes are crucial tasks. This abstract delves into the methodologies for accessing metadata such as column names, data types, and schema information, along with strategies for manipulating DataFrame data. The document explores techniques for retrieving DataFrame schema using the 'printSchema()'method and accessing column names and data types using attributes like 'schema', 'columns', and 'dtypes'. It also discusses advanced operations such as renaming columns, changing data types, adding new columns, and dropping columns using methods like 'withColumnRenamed()', 'cast()', 'withColumn()', and 'drop()'. Through these methods, data engineers and analysts can effectively manage and transform DataFrame metadata, enhancing data processing workflows and facilitating insightful data analysis.

INTRODUCTION:

Apache Spark is a powerful framework for distributed data processing, offering extensive capabilities for working with structured data through its Data Frame API. One crucial aspect of data processing is managing Data Frame metadata, including column names, data types, and schema information. This report provides an overview of how to access and manipulate Data Frame metadata in Apache Spark using various methods from the Spark SQL and Data Frame API.

FEATURES OF APACHE SPARK

Apache Spark has following features.

- Speed − Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.
- Supports multiple languages − Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.
- Advanced Analytics − Spark not only supports ‘Map’ and ‘reduce’. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

SPARK DATAFRAME

In Spark, DataFrames are the distributed collections of data, organized into rows and columns. Each column in a DataFrame has a name and an associated type.

DataFrames are similar to traditional database tables, which are structured and concise. We can say that DataFrames are relational databases with better optimization techniques.

Spark DataFrames can be created from various sources, such as Hive tables, log tables, external databases, or the existing RDDs. DataFrames allow the processing of huge amounts of data.

WHY DATAFRAME?

When there is not much storage space in memory or on disk, RDDs do not function properly as they get exhausted. Besides, Spark RDDs do not have the concept of schema—the structure of a database that defines its objects. RDDs store both structured and unstructured data together, which is not very efficient.

RDDs cannot modify the system in such a way that it runs more efficiently. RDDs do not allow us to debug errors during the runtime. They store the data as a collection of Java objects.

RDDs use serialization (converting an object into a stream of bytes to allow faster processing) and garbage collection (an automatic memory management technique that detects unused objects and frees them from memory) techniques. This increases the overhead on the memory of the system as they are very lengthy.

This was when DataFrames were introduced to overcome the limitations Spark RDDs had. Now, what makes Spark DataFrames so unique? Let’s check out the features of Spark DataFrames that make them so popular.

STREAMING IN SPARK:

- Streaming in Apache Spark refers to the real-time processing of continuously flowing data, enabling the analysis of data as it arrives or

changes over time. Spark Streaming, a component of the Apache Spark framework, provides support for processing live data streams with high throughput and fault tolerance.

- It allows developers to write streaming applications using familiar batch processing APIs, enabling seamless integration with existing Spark workflows.
- Spark Streaming ingests data from various sources such as Kafka, Flume, or TCP sockets, processes it using Spark's powerful distributed computing engine, and produces results in near real-time.

SERIALIZATION IN SPARK:

Serialization in Apache Spark refers to the process of converting data objects into a format suitable for transmission or storage, typically for distributed computing purposes.Format such as json, avro, csv,text etc. Spark uses serialization to efficiently transfer data between nodes in a distributed cluster and to persist data to disk or external storage systems.

PROJECT WORK FLOW:

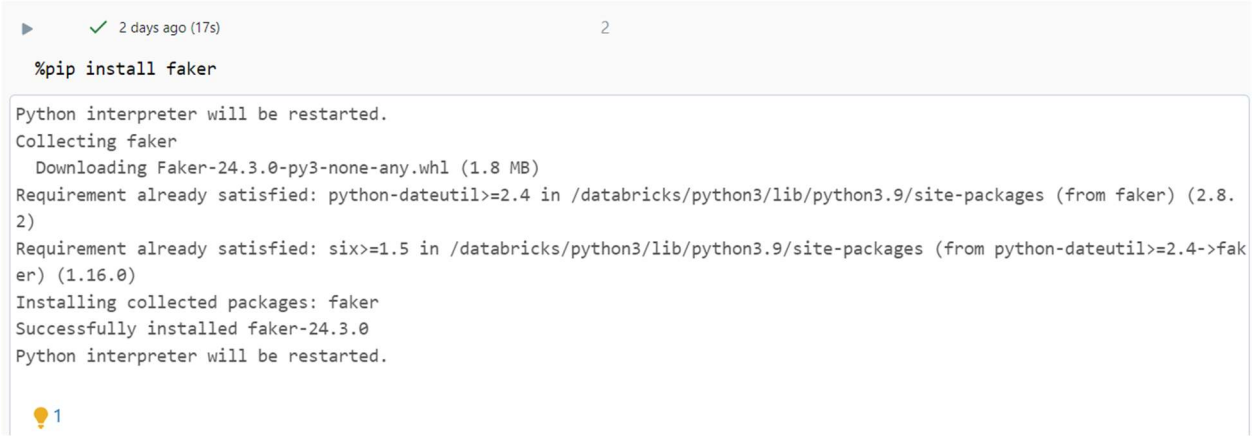
- Generate fake dataset by importing faker().
- Data cleaning- Removing unnecessary columns and removing special characters from column.
- Converting “duration” column from minutes to minutes-seconds using UDF.
- Converting “bpm” column from minutes to seconds using UDF.
- Merge the date, month and year column into single column as “date” using UDF.
- Load the dataframe into Google Cloud Bucket by creating Bucket instance and perform serialization.

❖ GENERATE DATA

Using Python, I generated the data for accessing and manipulating the Data Frame

Step 1:

The **%pip** magic command allows you to install packages from PyPI (Python Package Index) directly within your Databricks environment. Once installed, you can use the **faker** package to generate fake data for testing or simulation purposes.



Step 2:

I have imported several libraries and modules in Python for data manipulation and visualization using Spark Data Frame (‘pyspark.sql’), Faker (‘faker’). Each of these libraries serves a specific purpose in your data analysis workflow. Here's a brief explanation of each import statement and its role:

1. Import Spark Session and Data Types from PySpark:

- **‘from pyspark.sql import SparkSession’:**

Imports the ‘SparkSession’ class from the ‘pyspark.sql’ module, which is used to interact with Spark SQL and create DataFrame objects.

- **‘from pyspark.sql. types import StructType, StructField, StringType, Integer Type, Timestamp Type, Array Type’:**

Imports various data types and structures from ‘pyspark.sql.types’ module, such as

- **StructType** - StructType is a class used to define the structure of a DataFrame schema. It represents a collection of StructField objects that define the columns and their data types.
- **StructField** - StructField is used within StructType to define individual columns of a DataFrame schema. It specifies the name, data type, and nullable property of each column.
- **StringType** - StringType is a data type used to represent string values in Spark Data Frames. It is used for columns containing text or alphanumeric data.
- **IntegerType** - IntegerType is a data type used to represent integer values in Spark DataFrames. It is used for columns containing whole numbers.
- **FloatType-** FloatType is a data type used to represent float values in Spark DataFrames. It is used for columns containing collections of values.
- **BooleanType** - BooleanType is a data type used to represent Boolean values in Spark DataFrames. It is used for columns containing collections of values.

which are commonly used for defining schema structures for Spark DataFrames.

2. Import DataFrame Functions from PySpark:

- **‘from pyspark.sql.Functions import col, date_format, to_date, to_timestamp’:**

Imports DataFrame functions from ‘pyspark.sql. functions’ module, including ‘col ()’ for column selection, date formatting functions like ‘udf’, ‘regexp_replace’,’col,concat’,’date_format’,’lit’,’udf’,’regexp_replace’,’concat’,’d a te_format’.

3. Import Aggregate Functions and Others from PySpark:

- ‘from pyspark.sql.functions import col lit’: Imports additional DataFrame functions like ‘lit()’ for creating literal values.

4. Import Faker Library:

- ‘from faker import Faker’:

Imports the ‘Faker’ class from the ‘faker’ library, which is used for generating fake data such as names, addresses, emails, etc. This is commonly used for testing and simulation purposes.

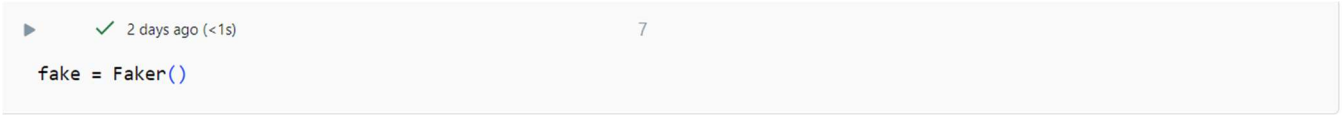
By importing these libraries and modules, you have access to a wide range of functionalities for working with data in Spark DataFrames, generating synthetic data with Faker, performing data analysis with Pandas, and creating visualizations using Matplotlib.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, TimestampType, ArrayType
from pyspark.sql.functions import col, date_format, to_date, to_timestamp
from pyspark.sql.functions import col, unix_timestamp, from_unixtime,sum,lit

from faker import Faker
import random
import pandas as pd
import matplotlib.pyplot as plt
```

Step: 3

This initializes an instance of the Faker class from the faker library in Python. The faker library is commonly used for generating fake data, such as names, addresses,

phone numbers, dates, and more, which can be useful for testing, prototyping, and generating sample datasets.



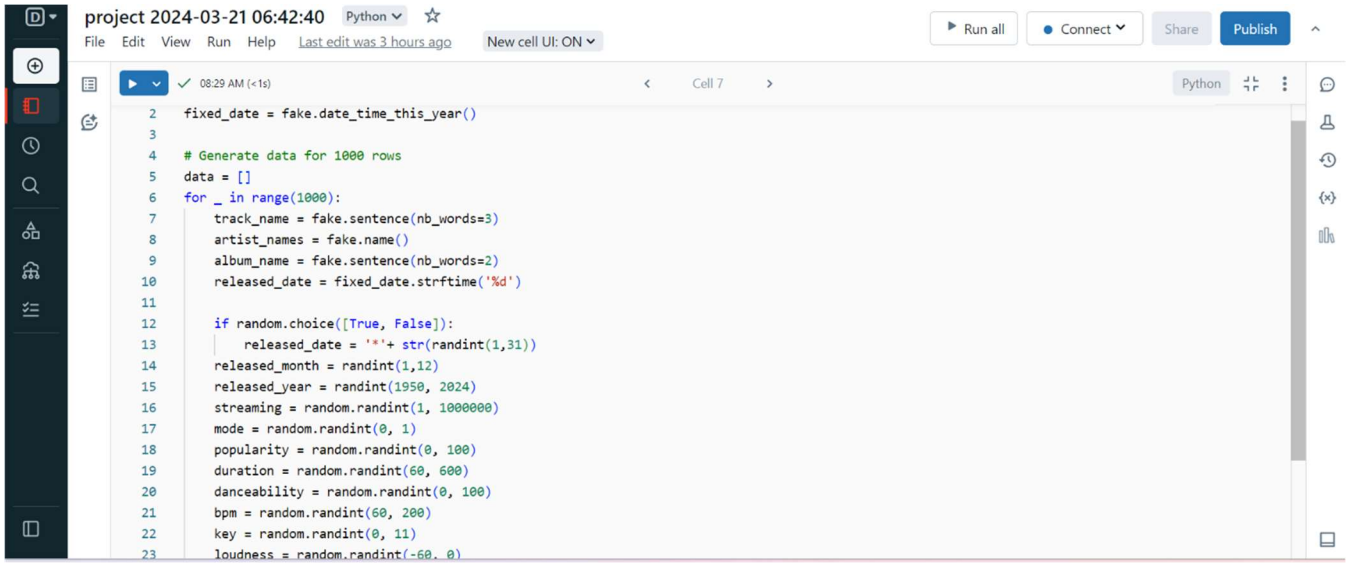
Step: 4

'date_time_this_year function generates fake spotify data based on the specified number of rows (num_rows). It uses the random module for generating random numbers and the faker library for generating fake text, artist name, track name, album name, streaming, year, month, date, bpm, duration, dancibility.

Here's an explanation of each component in the function:

1. **Fake word and text:**
    - Track Name Generation: Uses the fake.sentence() function to generate a fake track name consisting of three words.
    - Artist Name Generation: Generates a fake artist name using the fake.name() function.
2. **Released date, month, year:**
    - Generates a fixed day of the month (released_date) based on a predetermined date.
    - Randomly appends '*' to the released_date if a random choice evaluates to True.
    - Generates a random month (released_month) between 1 and 12.
    - Generates a random year (released_year) between 1950 and the current year (2024).

3. **Streaming:**
    - Streaming Count: Simulates the number of times the track has been streamed, ranging from 1 to 1,000,000.
4. **Mode:** Indicates the mode of the track (major or minor) represented by a binary value (0 or 1).
5. **Popularity:** Represents the popularity of the track as a random integer between 0 and 100.
6. **Duration:** Specifies the duration of the track in seconds, ranging from 60 to 600 seconds.
7. **Danceability:** Assigns a danceability score to the track, ranging from 0 to 100.
8. Beats Per Minute (BPM): Sets the tempo of the track in beats per minute, ranging from 60 to 200.
9. **Key:** Specifies the musical key of the track, represented by a random integer between 0 and 11.
10. **Loudness:** Sets the loudness level of the track in decibels, ranging from -60 dB to 0 dB.



‘the date_time function generate (1000)’ function call generates a list of 1000 tuples, with each tuple representing a piece of fake spotify data. You can adjust the number of rows (num_rows) as needed to generate more or fewer data entries.

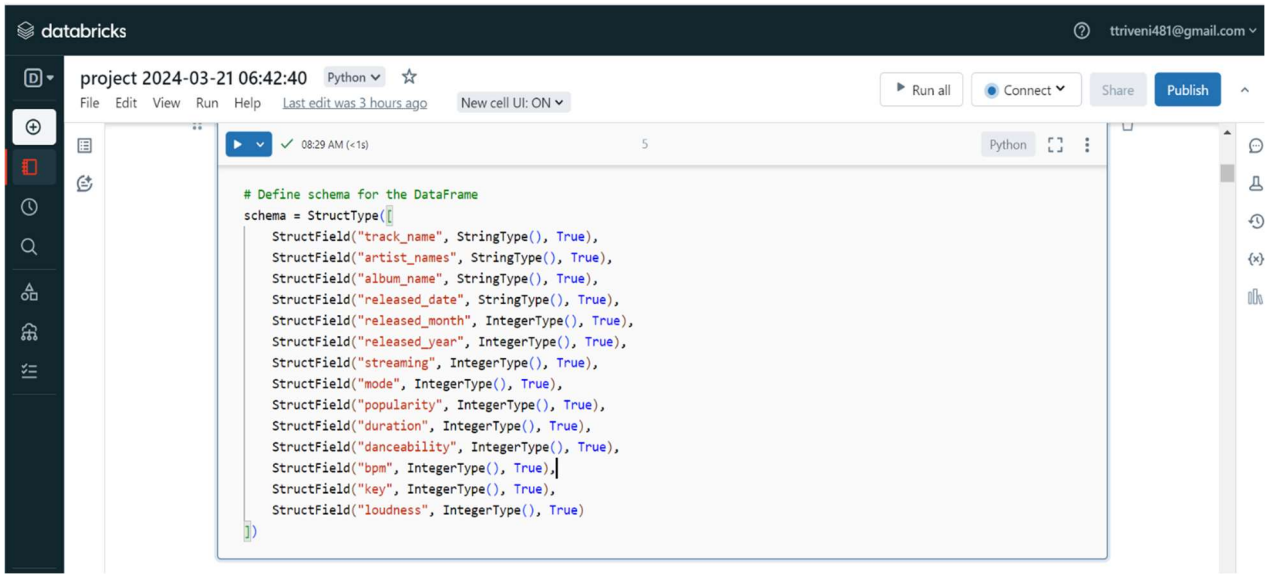
Step: 5

Defining a schema using the StructType and StructField classes from the pyspark.sql.types module in PySpark. This schema is used to define the structure and data types of the columns for a DataFrame in PySpark.

Here's an explanation of each field in the schema:

1. **track name"** (StringType): Represents the name of the music track.
2. **"artist names"** (StringType): Represents the name of the artist or artists associated with the track.
3. **"album name"** (StringType): Represents the name of the album to which the track belongs.
4. **"released date"** (StringType): Represents the day of the month when the track was released. This field is of StringType, indicating that the date is represented as a string.
5. **"released month"** (IntegerType): Represents the month when the track was released. This field is of IntegerType, indicating that the month is represented as an integer.
6. **"released year"** (IntegerType): Represents the year when the track was released. This field is of IntegerType, indicating that the year is represented as an integer.
7. **"streaming"** (IntegerType): Represents the number of times the track has been streamed.

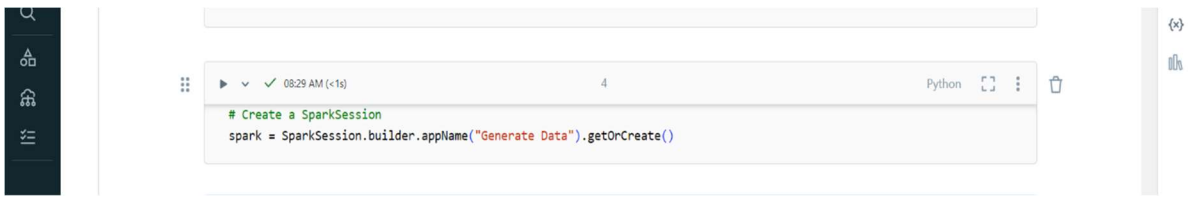
8. **"mode"** (IntegerType): Represents the mode of the track (major or minor), typically encoded as 0 for minor and 1 for major.
9. **"popularity"** (IntegerType): Represents the popularity score of the track.
10. **"duration"** (IntegerType): Represents the duration of the track in seconds.
11. **"danceability"** (IntegerType): Represents the danceability score of the track.
12. **"bpm"** (IntegerType): Represents the tempo of the track in beats per minute (BPM).
13. **"key"** (IntegerType): Represents the musical key of the track, typically encoded as an integer value between 0 and 11.
14. **"loudness"** (IntegerType): Represents the loudness level of the track in decibels (dB)



Step : 6

The line **spark=SparkSession.builder.appName("Generate Data").getOrCreate()** creates a Spark session named "Metadata" using the SparkSession builder in Apache Spark.

1. **SparkSession:**
    - SparkSession is the entry point to programming Spark with the Dataset and DataFrame API. It provides a unified interface for interacting with Spark functionality and allows you to work with structured data.
2. **builder:**
    - The builder method is used to create a Builder object for configuring and setting up the Spark session.
3. **appName("Generate Data"):**
    - The appName method sets the name of the Spark application to "Metadata". This name is displayed in the Spark UI and logs, making it easier to identify your application.
4. **getOrCreate ():**
    - The getOrCreate method checks if there is an existing Spark session available. If an active Spark session exists, it returns that session. Otherwise, it creates a new Spark session based on the configuration set using the builder object.

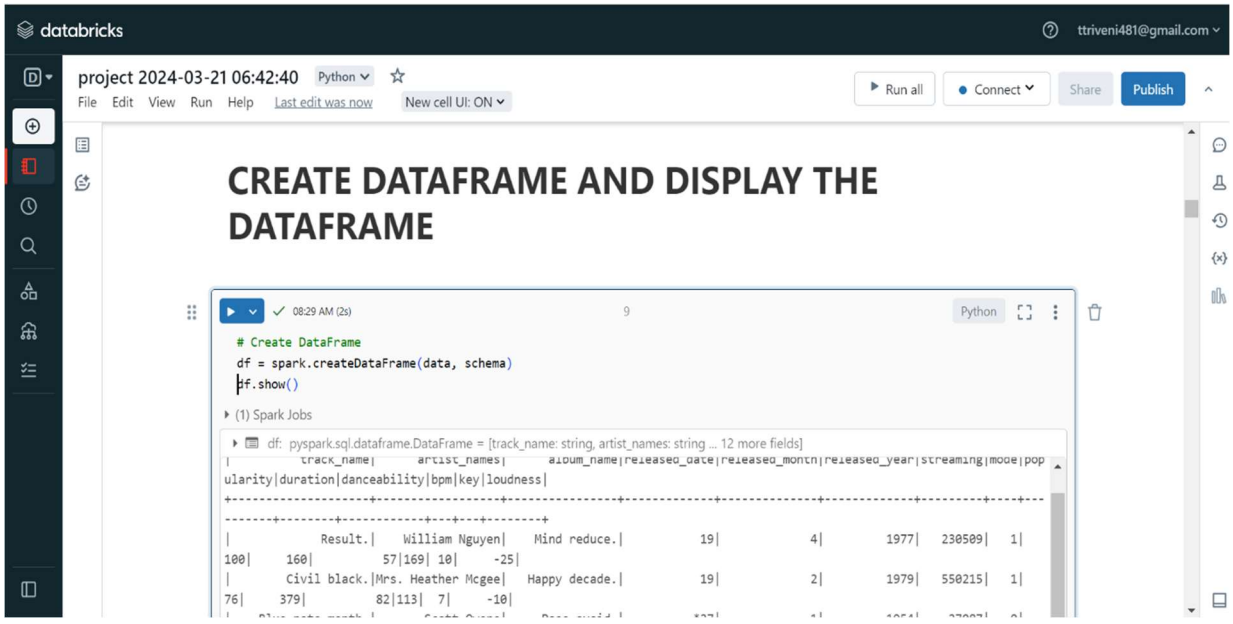


Step : 7

Creating a PySpark DataFrame named df using the provided spotify dataset and schema.

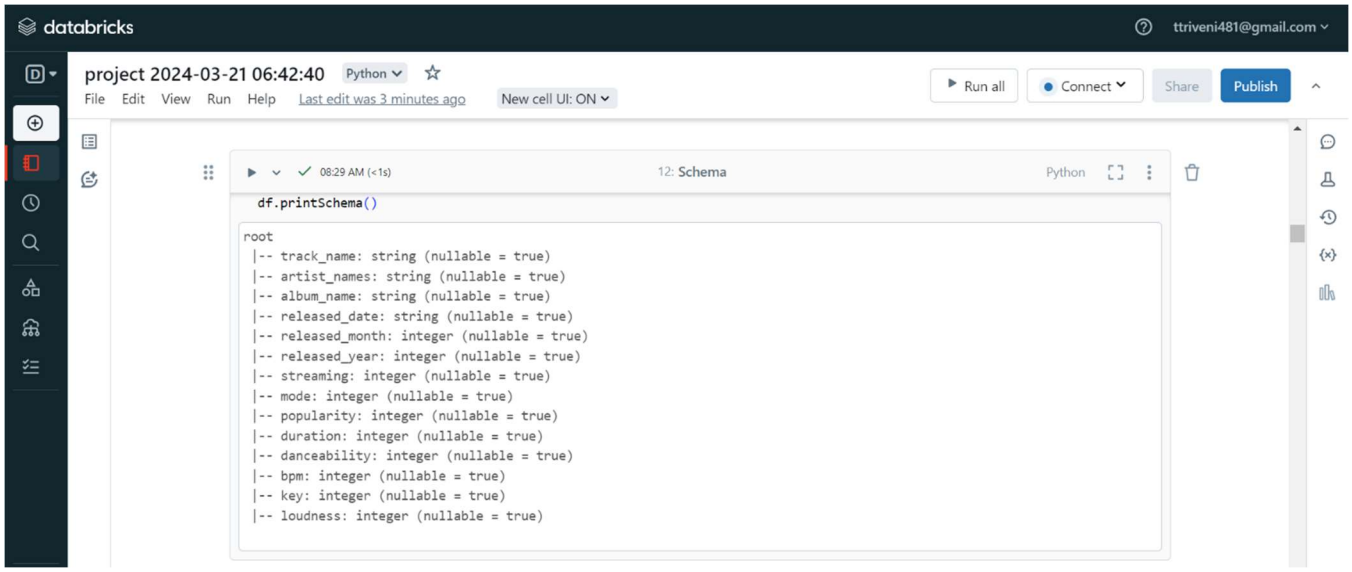
Here's how the code works:

1. spotify_data: This is assumed to be a list of tuples, where each tuple represents a row of data for the DataFrame. Each tuple should follow the structure defined by the schema you provided earlier.
2. schema: This is the schema that defines the structure of the DataFrame, including the names, data types, and nullable settings of each column.
3. spark.createDataFrame(data, schema): This method creates a PySpark DataFrame (df) using the provided data (data) and schema (schema). The data is mapped to the columns defined in the schema, and the DataFrame is created accordingly.



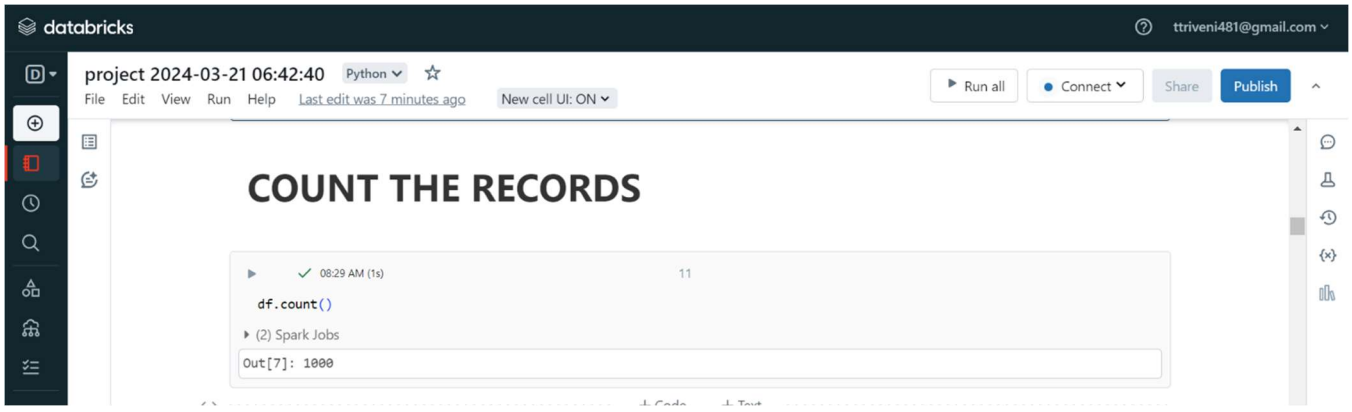
Step:8

To print the schema of a PySpark DataFrame before performing any manipulations, you can use the printSchema() method on the DataFrame object. it will print the schema of the Data Frame- df to the console or output window. The schema will include the names, data types of each column in the DataFrame**.**



Step: 9

Using the count function, total records of the dataframe will be showed.



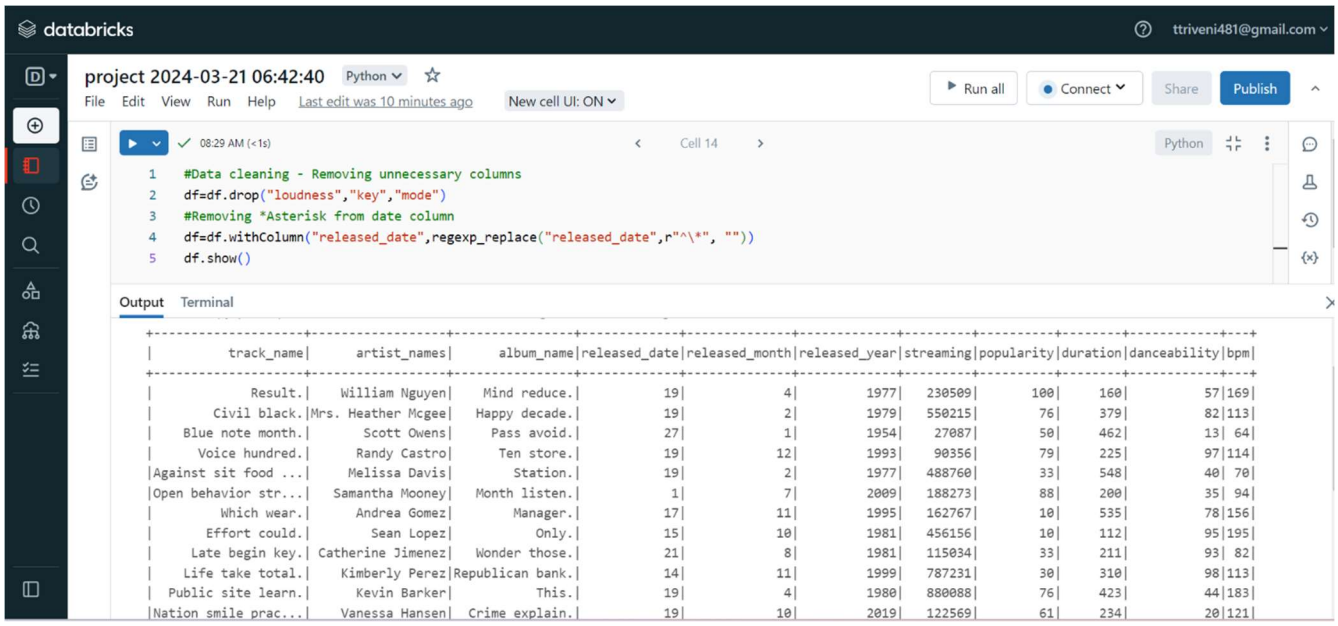
Step: 10

Handle missing or null values by imputing them with appropriate values, dropping unnecessary rows or columns. Removing special characters from column.

'drop': The drop function is used to remove one or more columns from a DataFrame. It takes column names as arguments and returns a new DataFrame with the specified columns removed.

'withColumn': The withColumn function is used to add a new column to a DataFrame or replace an existing column with a modified version. It takes a column name and an expression as arguments, where the expression computes the values for the new column based on the existing columns.

'regexp': regexp is not a function in Spark directly. However, regular expressions (regex) can be used with various functions like filter, where, select, like, rlike, etc., for pattern matching and string manipulation.



track_name	artist_names	album_name	released_date	released_month	released_year	streaming	popularity	duration	danceability	bpm
Result.	William Nguyen	Mind reduce.	19	4	1977	230509	100	160	57	169
Civil black.	Mrs. Heather Mcgee	Happy decade.	19	2	1979	550215	76	379	82	113
Blue note month.	Scott Owens	Pass avoid.	27	1	1954	27087	50	462	13	64
Voice hundred.	Randy Castro	Ten store.	19	12	1993	90356	79	225	97	114
Against sit food ...	Melissa Davis	Station.	19	2	1977	488760	33	548	40	70
Open behavior str...	Samantha Mooney	Month listen.	1	7	2009	188273	88	200	35	94
Which wear.	Andrea Gomez	Manager.	17	11	1995	162767	10	535	78	156
Effort could.	Sean Lopez	Only.	15	10	1981	456156	10	112	95	195
Late begin key.	Catherine Jimenez	Wonder those.	21	8	1981	115034	33	211	93	82
Life take total.	Kimberly Perez	Republican bank.	14	11	1999	787231	30	310	98	113
Public site learn.	Kevin Barker	This.	19	4	1980	880088	76	423	44	183
Nation smile prac...	Vanessa Hansen	Crime explain.	19	10	2019	122569	61	234	20	121

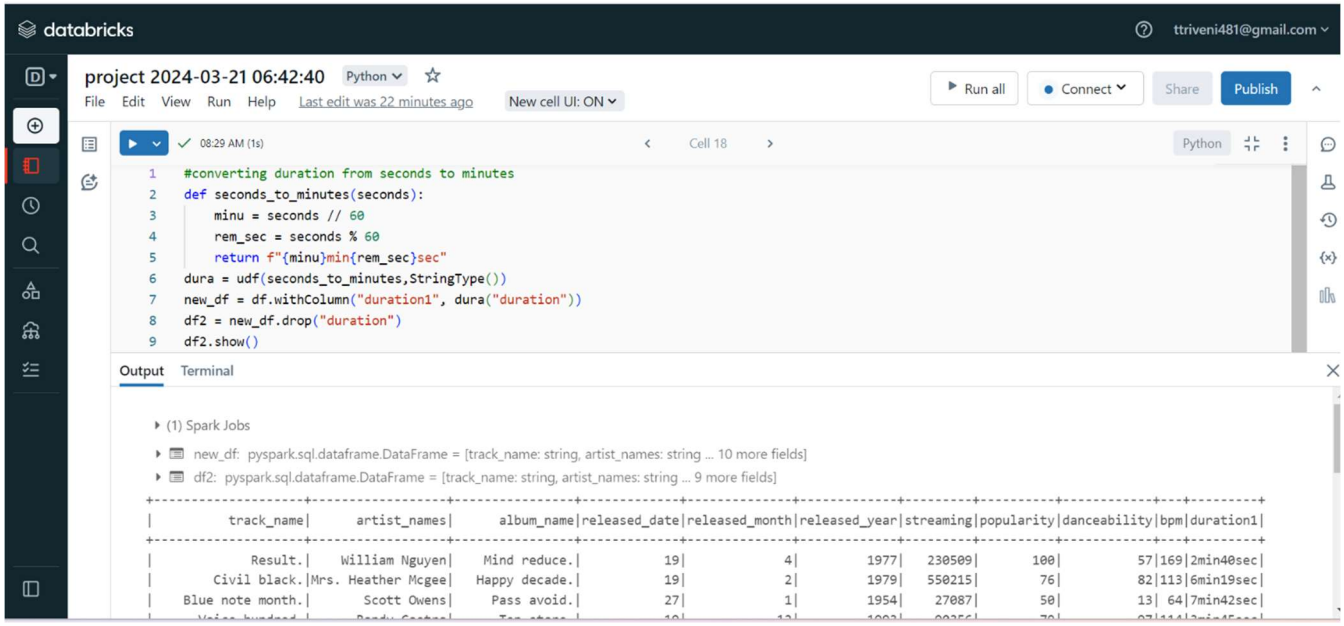
Step: 11

Converting the duration column from minutes to minutes -seconds by performing UDF functions.

1. **Function Definition (seconds_to_minutes):**
   - Defines a Python function named seconds_to_minutes that takes a number of seconds as input and returns a string representing the duration in minutes and seconds.
   - Inside the function, it calculates the number of minutes (minu) and the remaining seconds (rem_sec) after converting the input seconds to minutes.
   - Returns a formatted string representing the duration in the format "XminYsec".
2. **User Defined Function (UDF) Creation (dura):**
   - Creates a User Defined Function (UDF) named dura using the seconds_to_minutes function defined above.
   - The UDF is defined to operate on the duration column of the DataFrame and convert each value from seconds to minutes using the seconds_to_minutes function.
   - Specifies the return type of the UDF as StringType().
3. **DataFrame Transformation (withColumn):**
   - Applies the UDF dura to the duration column of the DataFrame (df) using the withColumn function.
   - Creates a new DataFrame (new_df) with the modified duration column representing the duration in minutes and seconds.
4. **DataFrame Manipulation (drop):**

- Creates another new DataFrame (df2) by dropping the original duration column from new_df using the drop function.
- This operation effectively removes the original duration column, leaving only the modified duration column representing the duration in minutes and seconds.

5. **Displaying DataFrame (show):**
   - Displaying the dataframe.



Step: 12

Converting the bpm(Beat Per Minute) to seconds and make a new column name ‘bpm_sec’ and dropping the existing bpm column.

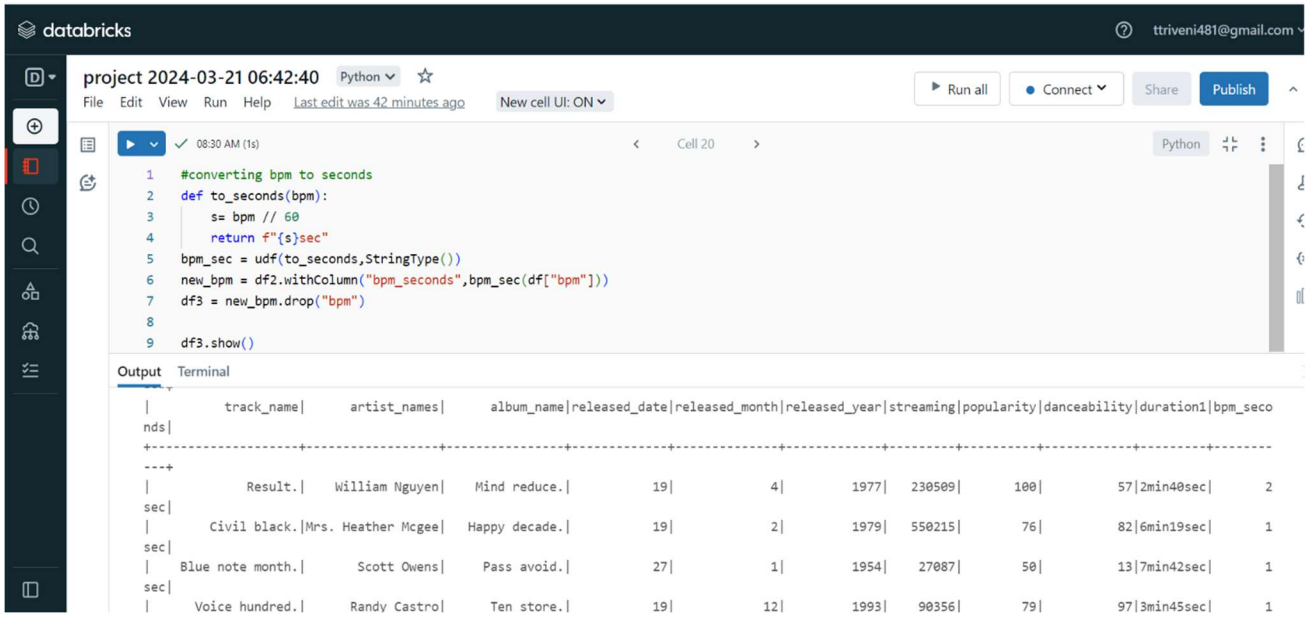
Function Definition (to_seconds):

- Defines a Python function named to_seconds that takes a BPM value as input and returns a string representing the BPM converted to beats per second.

- Inside the function, it calculates the number of beats per second (s) by dividing the BPM by 60 (since there are 60 seconds in a minute).
- Returns a formatted string representing the BPM in beats per second.

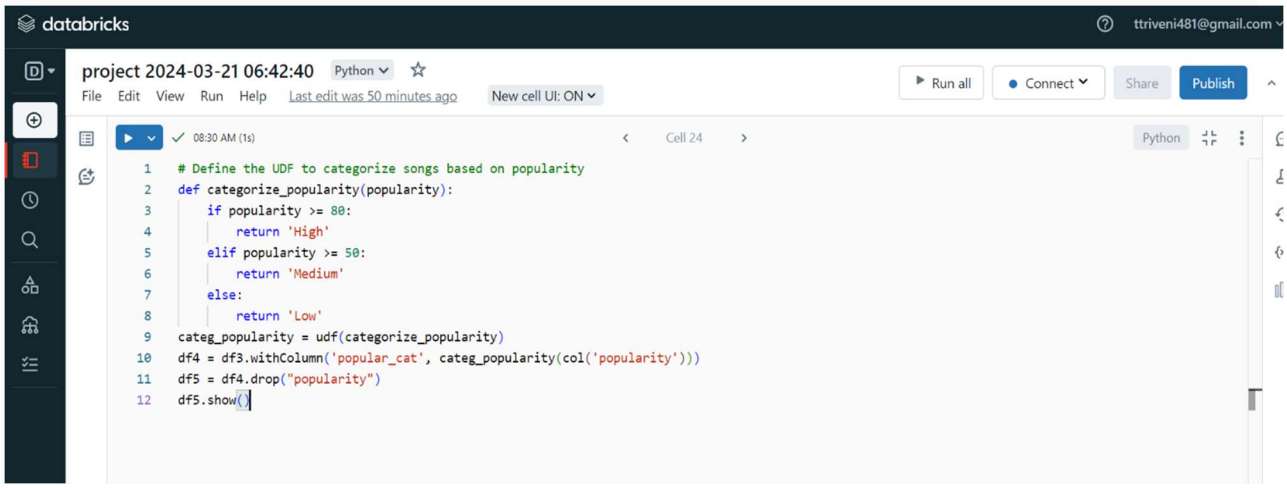
User Defined Function (UDF) Creation (bpm_sec):

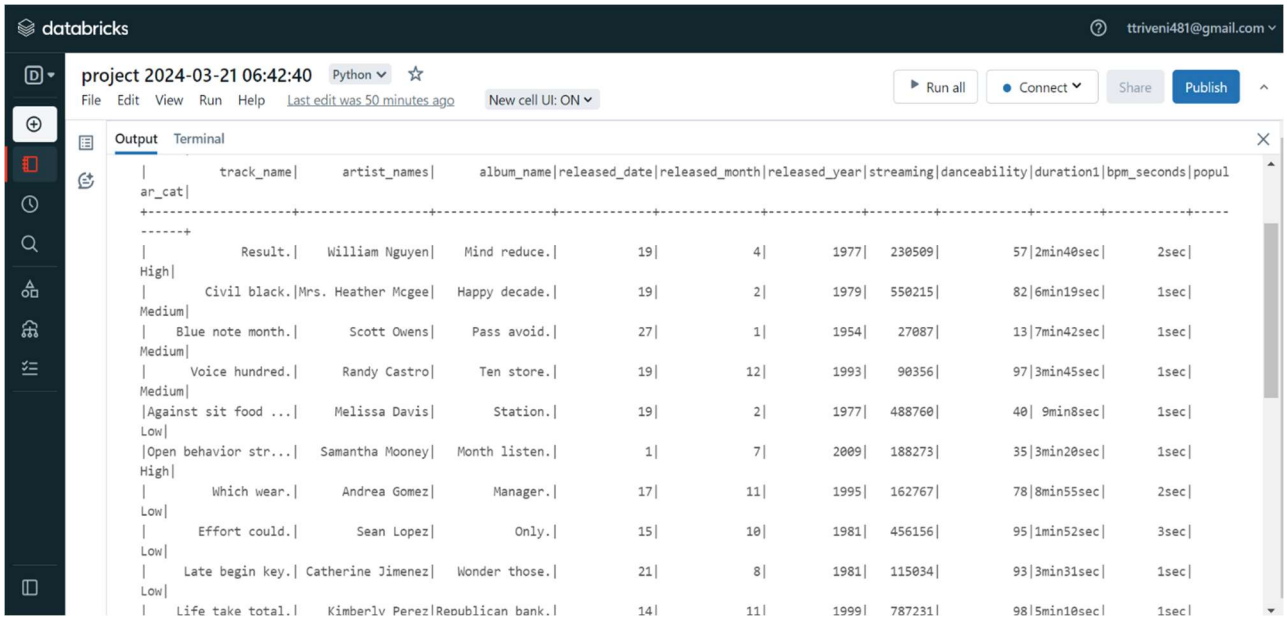
- Creates a User Defined Function (UDF) named bpm_sec using the to_seconds function.
- The UDF is defined to operate on the BPM column of the DataFrame and convert each value from beats per minute to beats per second using the to_seconds function.
- Specifies the return type of the UDF as StringType().



Step: 13

Define the UDF to categorize songs based on popularity and shows the dataframe.

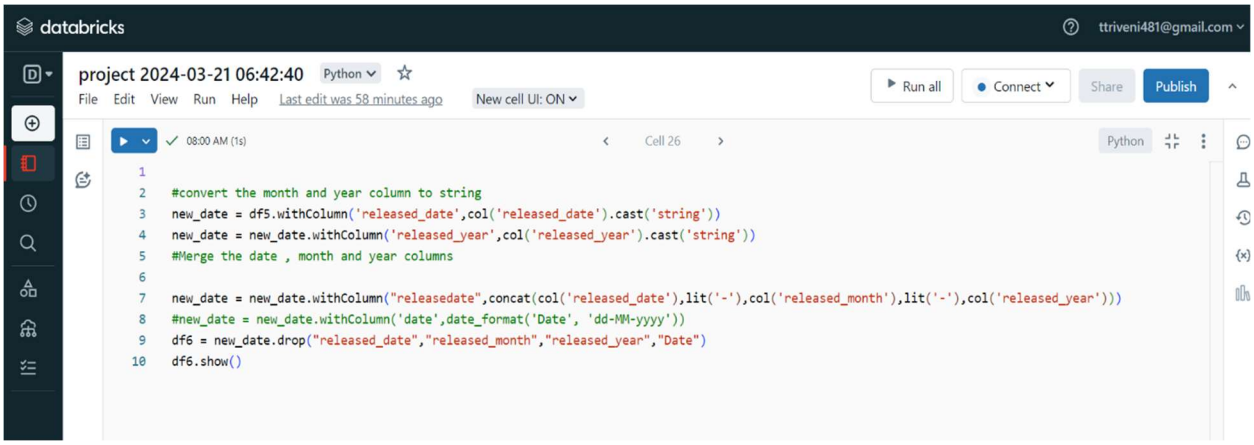


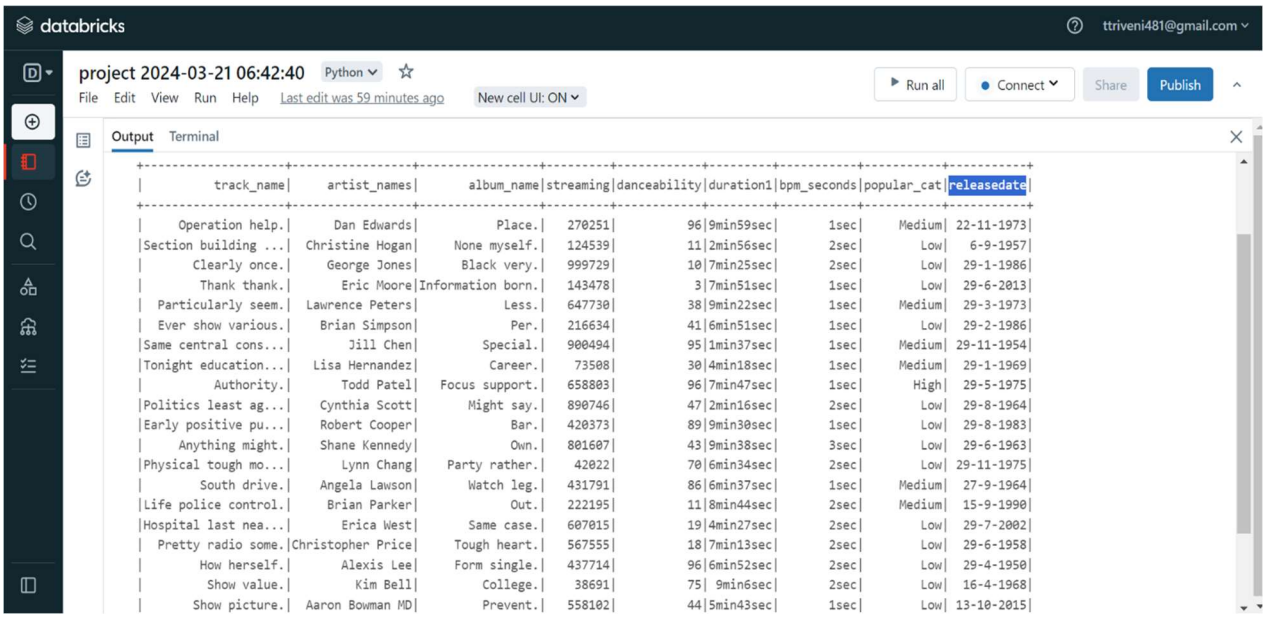


Step: 14

To convert the month and year to string by casting.

1. **'col_name().cast(Datatype)':**This command is used for convert datatype for the column.
2. **'concat' :** Using concat function merge the date, month, year column.

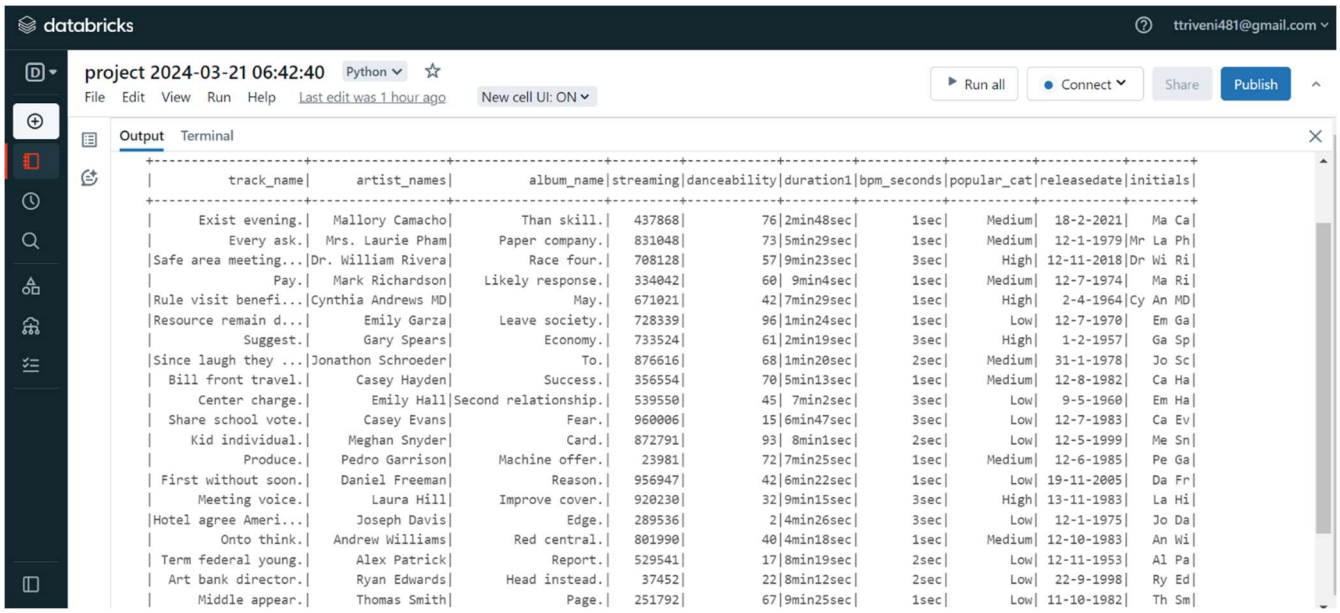




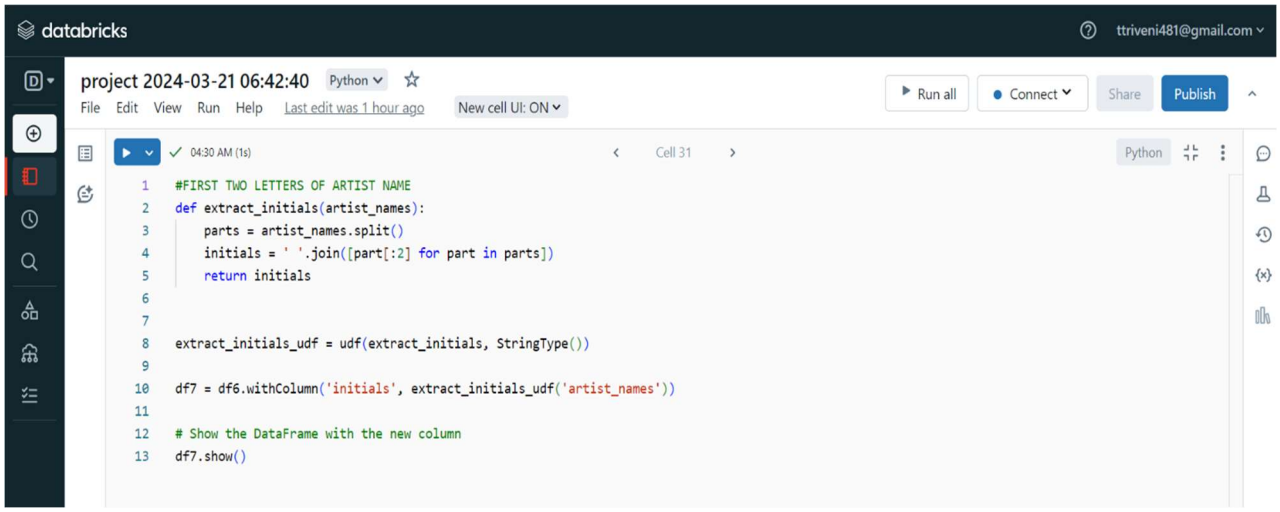
Step: 15

Using the withColumn method in PySpark to add columns in a DataFrame.

1. **‘df.withColumn ("Initials")’:**This creates the "Initials" column in the DataFrame df.



track_name	artist_names	album_name	streaming	danceability	duration1	bpm_seconds	popular_cat	releasedate	initials
Exist evening.	Mallory Camacho	Than skill.	437868	76	2min48sec	1sec	Medium	18-2-2021	Ma Ca
Every ask.	Mrs. Laurie Pham	Paper company.	831048	73	5min29sec	1sec	Medium	12-1-1979	Mr La Ph
Safe area meeting...	Dr. William Rivera	Race four.	708128	57	9min23sec	3sec	High	12-11-2018	Dr Wi Ri
Pay.	Mark Richardson	Likely response.	334042	60	9min4sec	1sec	Medium	12-7-1974	Ma Ri
Rule visit benefi...	Cynthia Andrews MD	May.	671021	42	7min29sec	1sec	High	2-4-1964	Cy An MD
Resource remain d...	Emily Garza	Leave society.	728339	96	1min24sec	1sec	Low	12-7-1970	Em Ga
Suggest.	Gary Spears	Economy.	733524	61	2min19sec	3sec	High	1-2-1957	Ga Sp
Since laugh they ...	Jonathon Schroeder	To.	876616	68	1min20sec	2sec	Medium	31-1-1978	Jo Sc
Bill front travel.	Casey Hayden	Success.	356554	70	5min13sec	1sec	Medium	12-8-1982	Ca Ha
Center charge.	Emily Hall	Second relationship.	539550	45	7min2sec	3sec	Low	9-5-1960	Em Ha
Share school vote.	Casey Evans	Fear.	960006	15	6min47sec	3sec	Low	12-7-1983	Ca Ev
Kid individual.	Meghan Snyder	Card.	872791	93	8min1sec	2sec	Low	12-5-1999	Me Sn
Produce.	Pedro Garrison	Machine offer.	23981	72	7min25sec	1sec	Medium	12-6-1985	Pe Ga
First without soon.	Daniel Freeman	Reason.	956947	42	6min22sec	1sec	Low	19-11-2005	Da Fr
Meeting voice.	Laura Hill	Improve cover.	920230	32	9min15sec	3sec	High	13-11-1983	La Hi
Hotel agree Ameri...	Joseph Davis	Edge.	289536	2	4min26sec	3sec	Low	12-1-1975	Jo Da
Onto think.	Andrew Williams	Red central.	801990	40	4min18sec	1sec	Medium	12-10-1983	An Wi
Term federal young.	Alex Patrick	Report.	529541	17	8min19sec	2sec	Low	12-11-1953	Al Pa
Art bank director.	Ryan Edwards	Head instead.	37452	22	8min12sec	2sec	Low	22-9-1998	Ry Ed
Middle appear.	Thomas Smith	Page.	251792	67	9min25sec	1sec	Low	11-10-1982	Th Sm



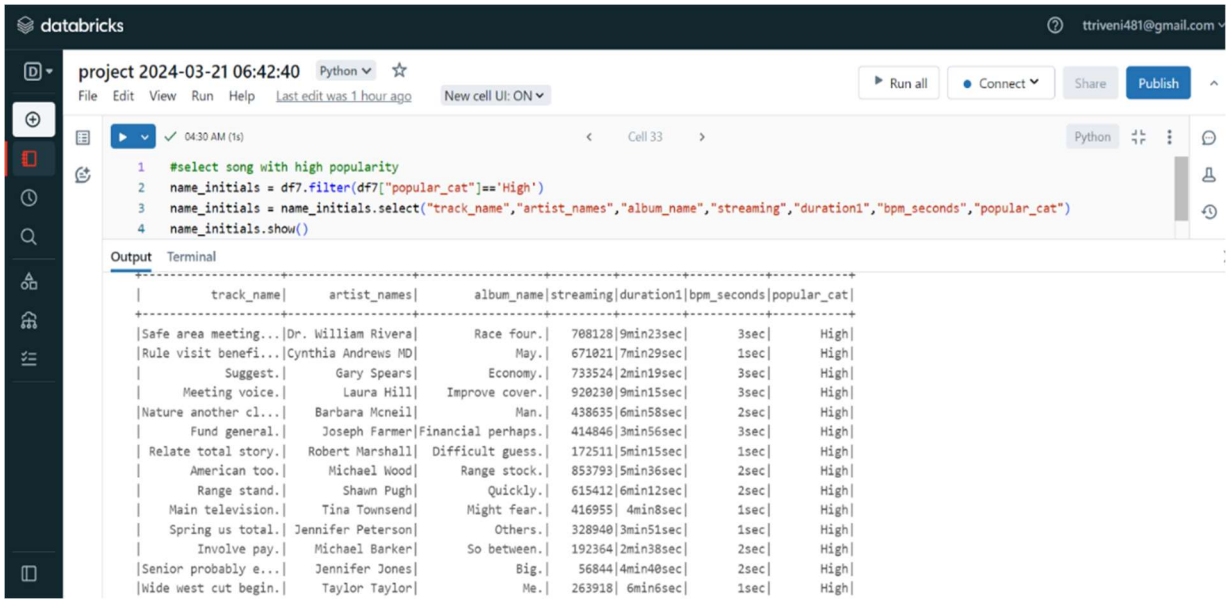

```python
#FIRST TWO LETTERS OF ARTIST NAME
def extract_initials(artist_names):
    parts = artist_names.split()
    initials = ' '.join([part[:2] for part in parts])
    return initials


extract_initials_udf = udf(extract_initials, StringType())

df7 = df6.withColumn('initials', extract_initials_udf('artist_names'))

# Show the DataFrame with the new column
df7.show()
```

Step 16:

Select the song with high popularity and also selecting the particular columns.

1.'select': This select function is used to selecting the particular columns from the dataframe.

2. 'Filter' : Filter function is used to filtering the column by the given condition.



Step: 17

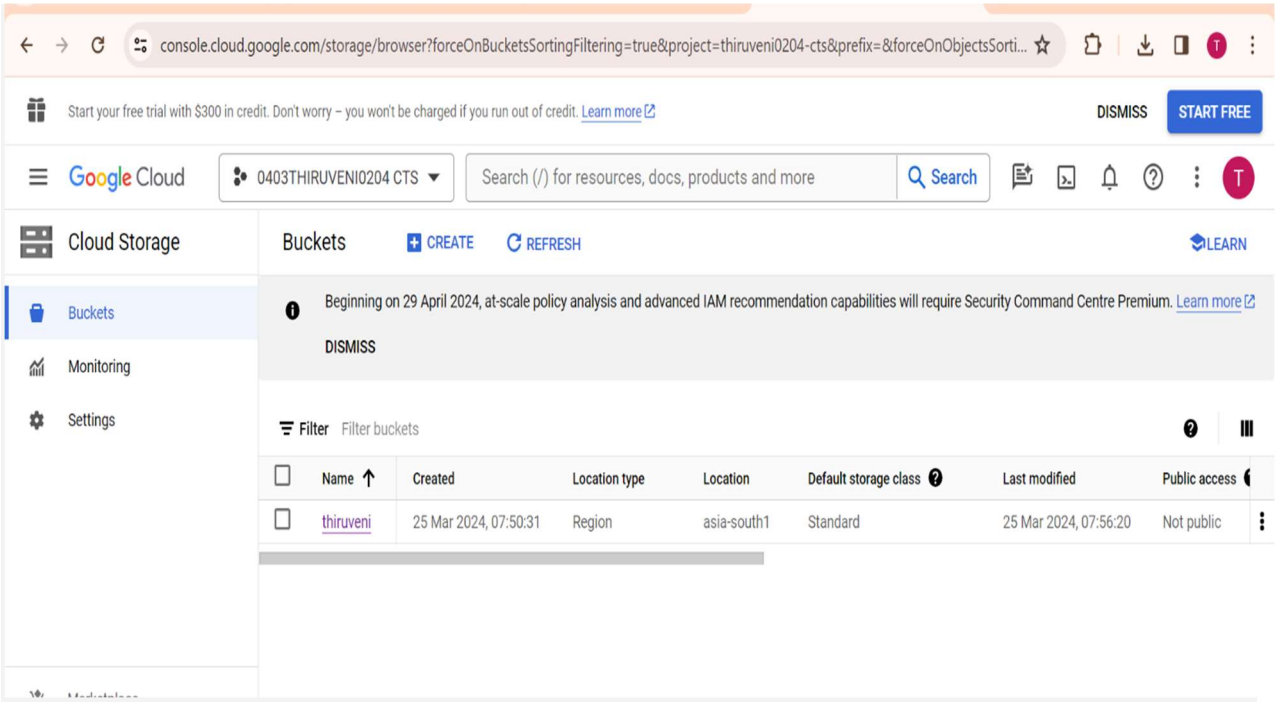
LOAD THE DF TO BUCKET

- Buckets are the basic containers that hold your data within Cloud Storage.
- Everything you store in Cloud Storage must be contained in a bucket.
- Bucket names are publicly visible, so avoid using personally identifiable information (PII) or sensitive data in them.

1.'Create Bucket in Google Cloud':

Cloud storage services allows users to create buckets to store their data. Each bucket typically has a unique name within the cloud storage.

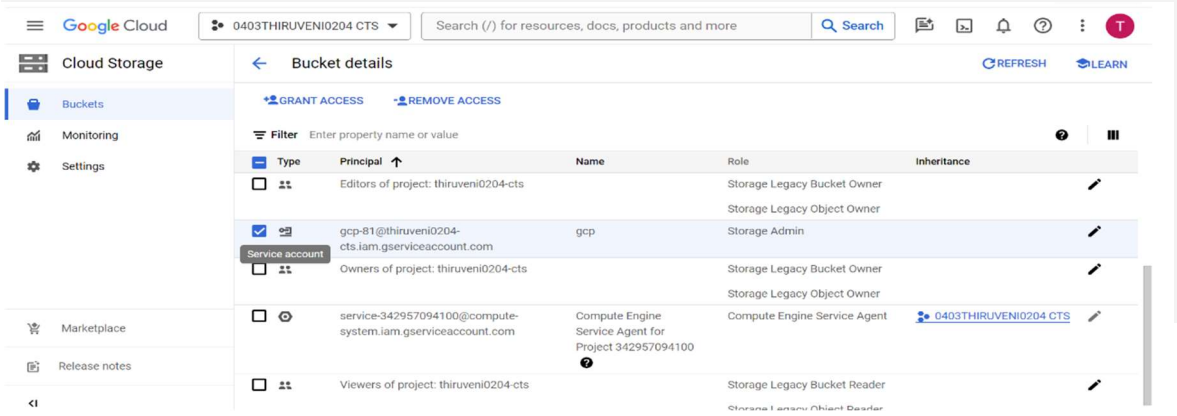
- Go to cloud storage – Click Bucket then click Create Bucket and give Bucket name
- Regions and zone (asia-Mumbai)
- Create bucket
- Now the created bucket is displayed in bucket page.



Step 18:

CREATING SERVICE ACCOUNT:

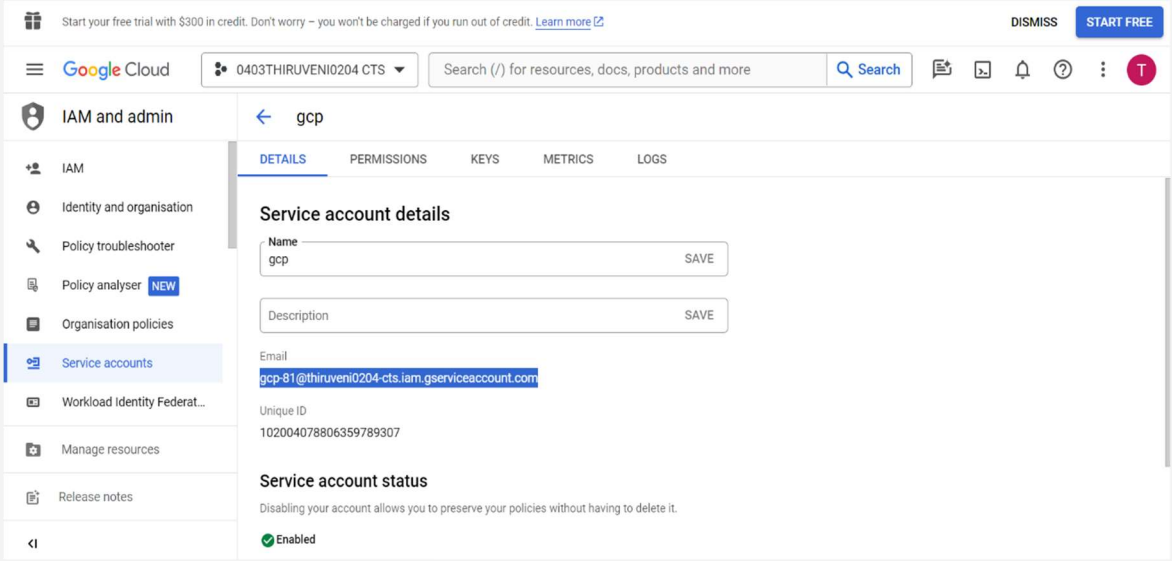
A service account represents google cloud identity such as code running on the compute engine VMs, App Engine running outside Google.



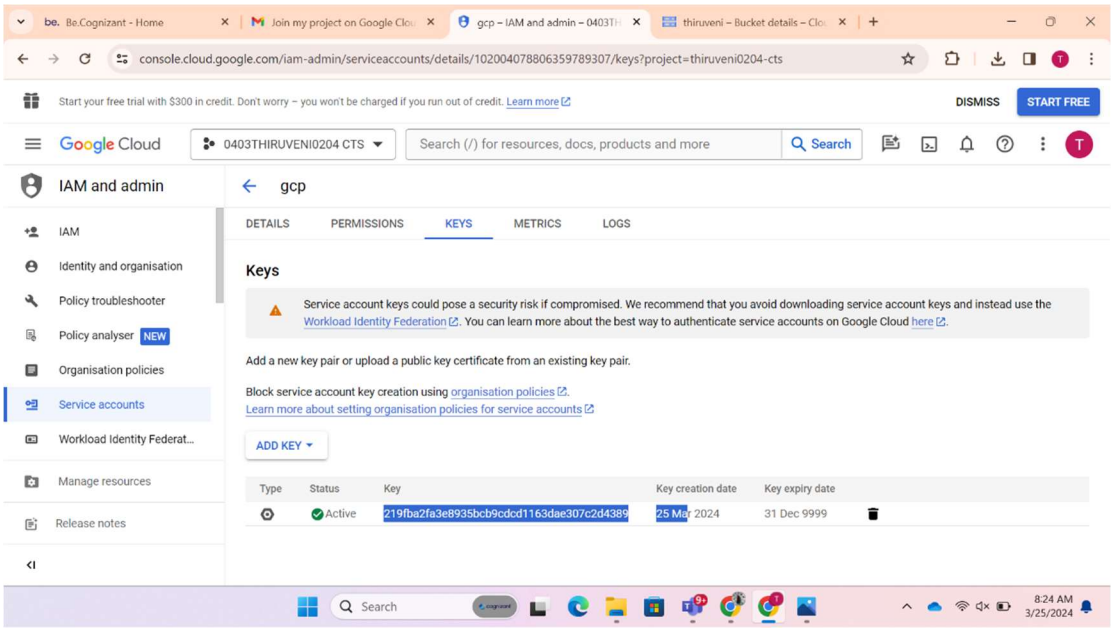
Step 19:

COPYING THE SERVICE ACCOUNT DETAILS

Go to service account and copy the email that was generated in the Details tab.

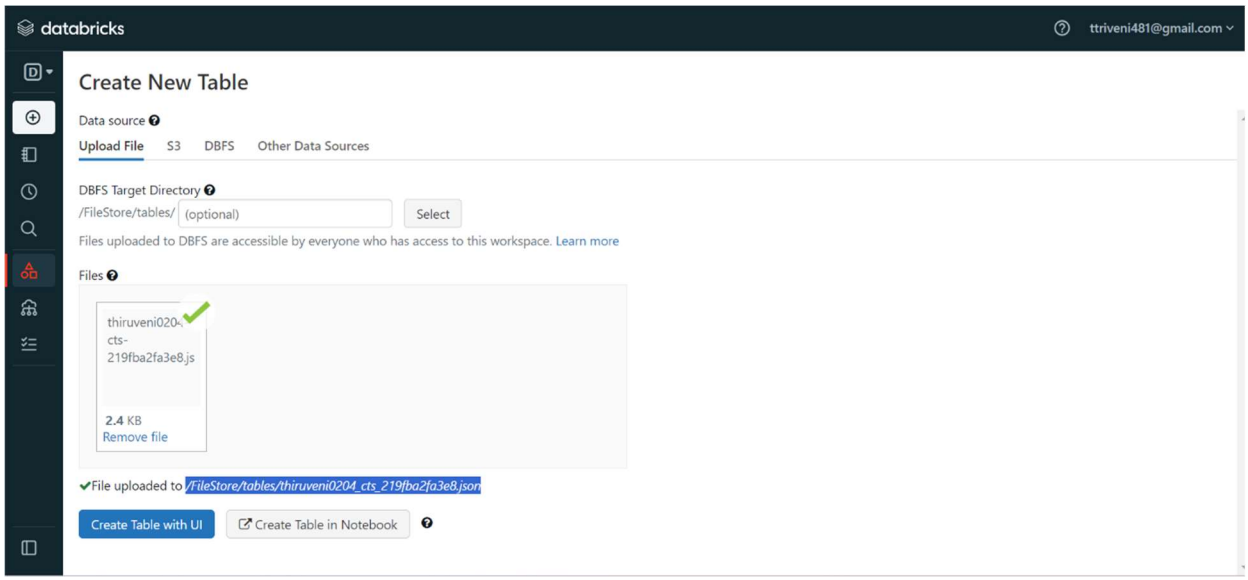


In key tab we have downlod JSON file that was stored in our local system.

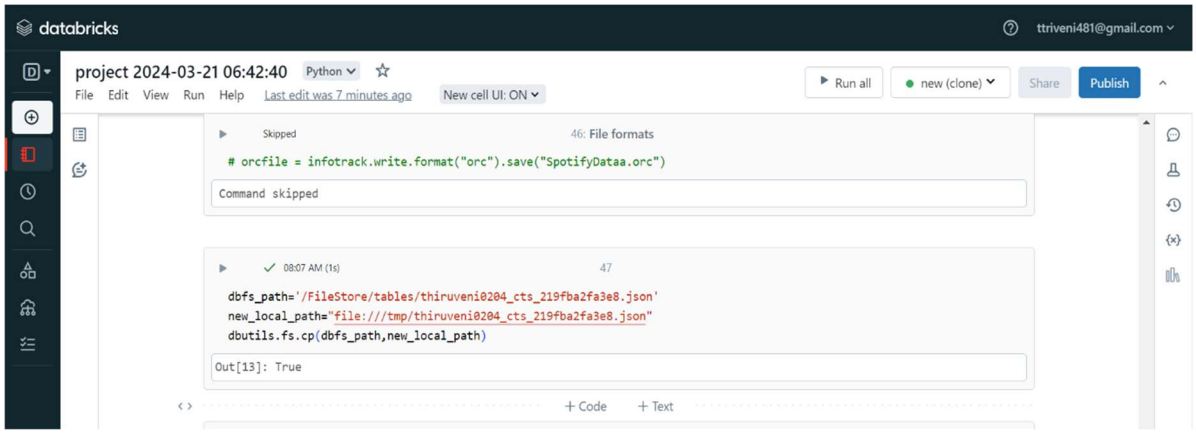


Step 20:

Upload the key JSON file in the Databricks DBFS.

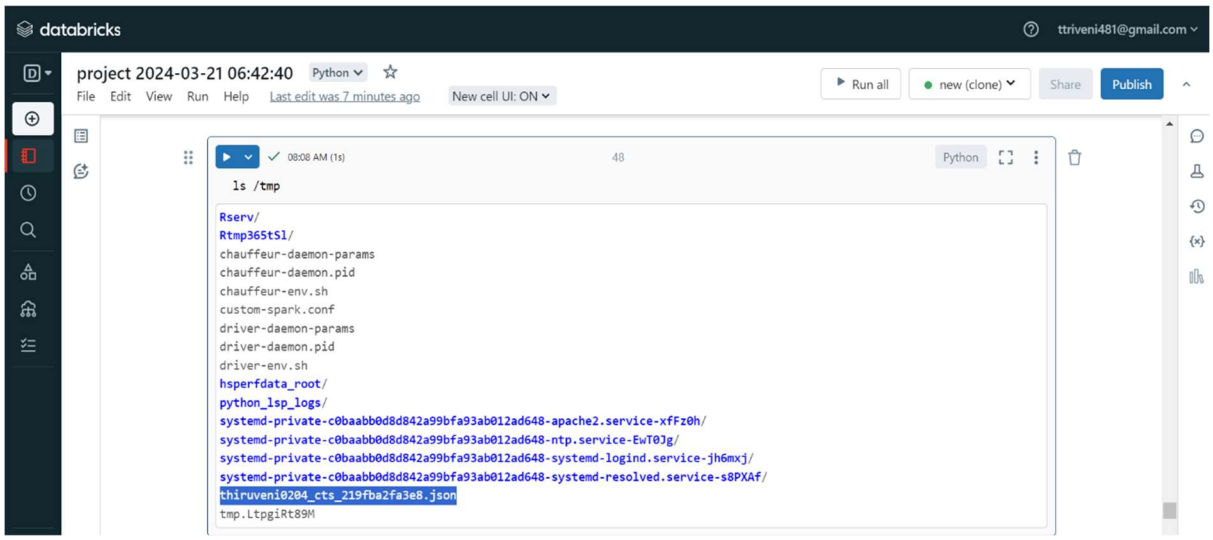


Create path from local to DBFS by uploading the key Json file in DBFS Databricks.

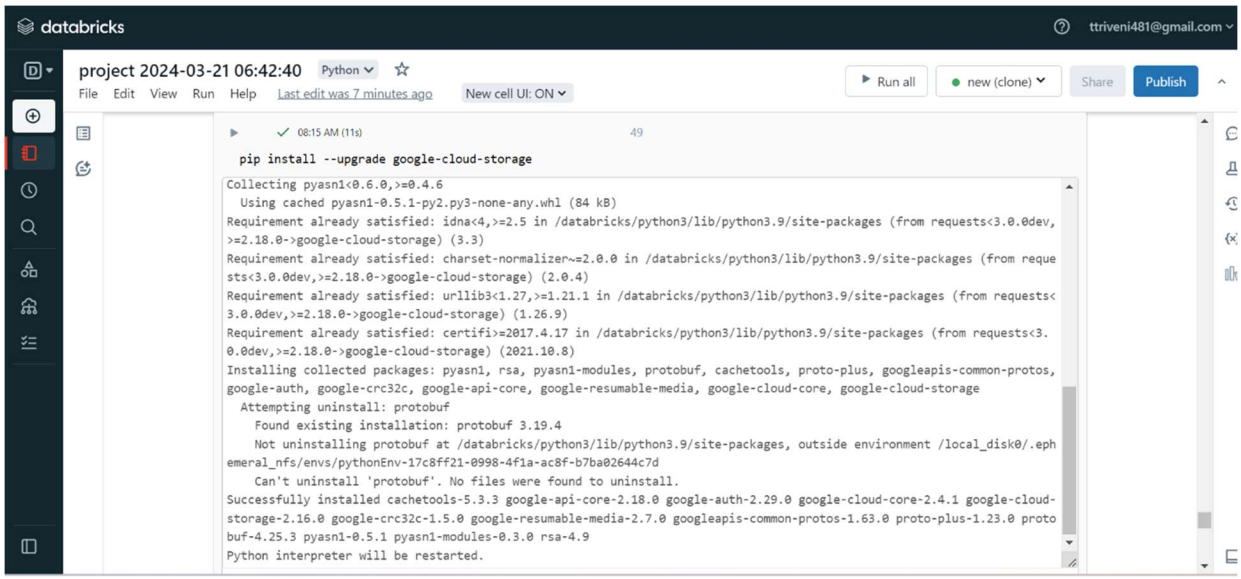


Step 21:

'LS' command is used to list the folders in the DBFS.



Install the Google Cloud storage in Databricks using pip command.

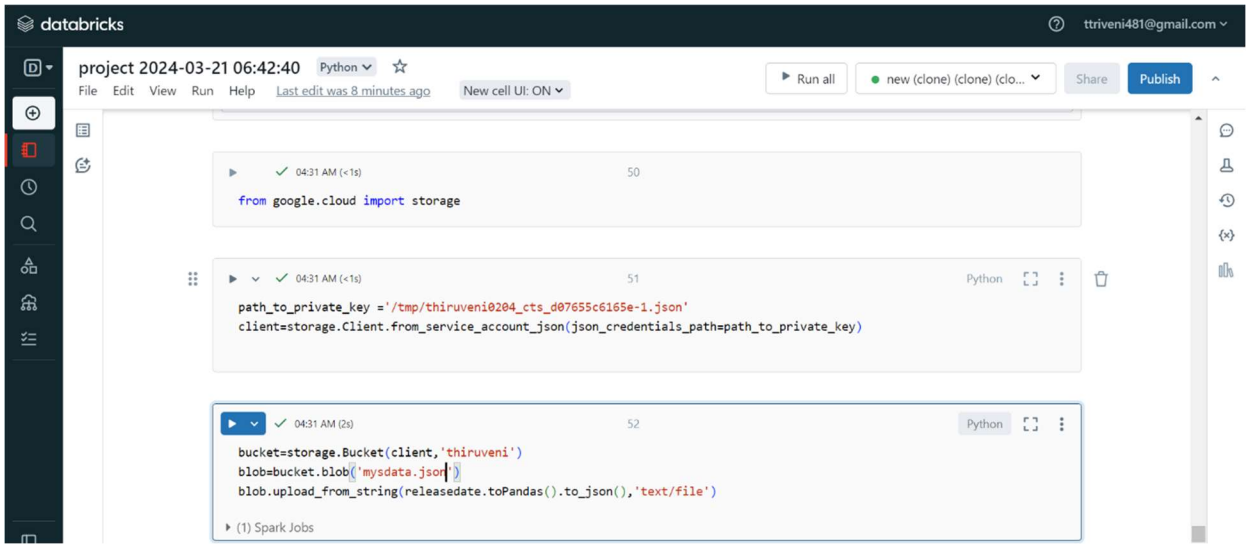


‘Import google cloud’ package in the Databricks.

Step 22:

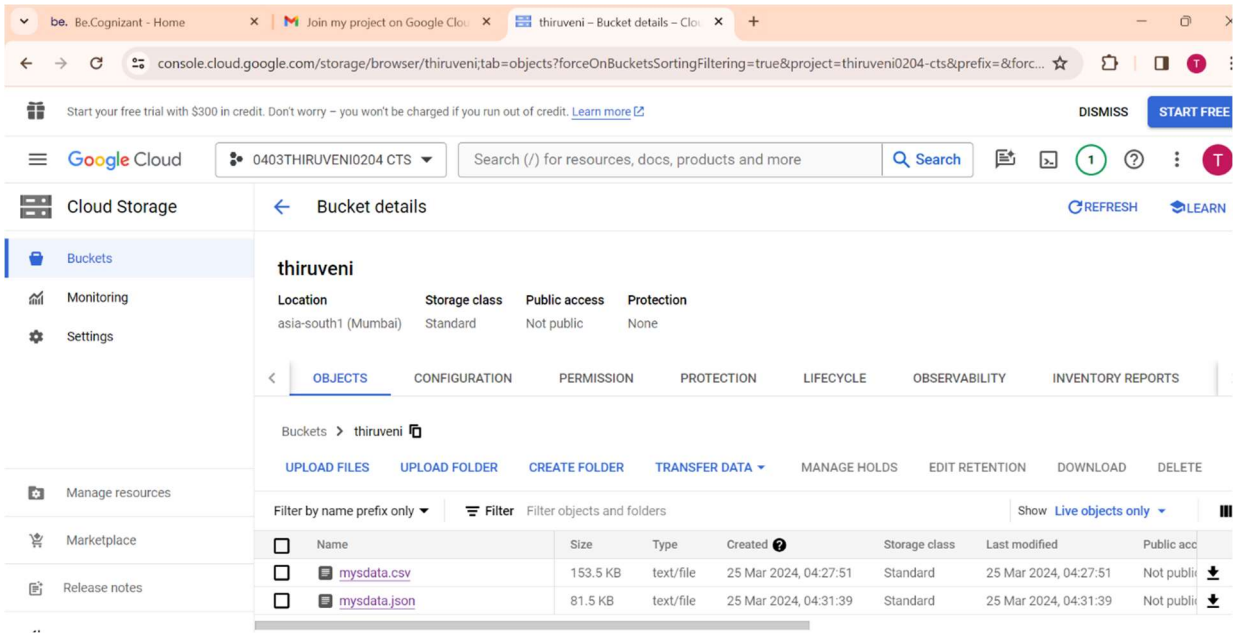
SERIALIZATION:

Loading my DBFS file to Google Cloud Bucket as **JSON file format**.

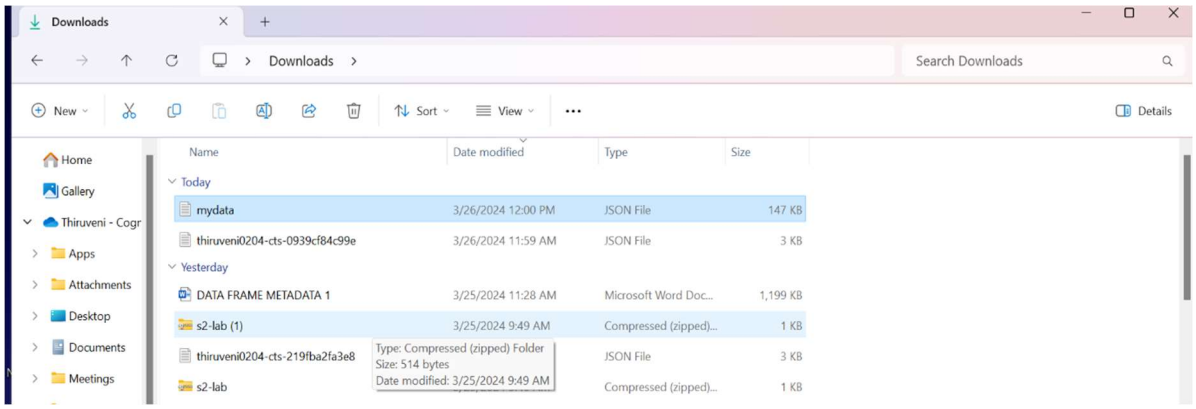


Loading my DBFS file to Google Cloud Bucket as **csv file format**.

Now JSON and CSV files are uploaded in the Google Cloud Bucket.



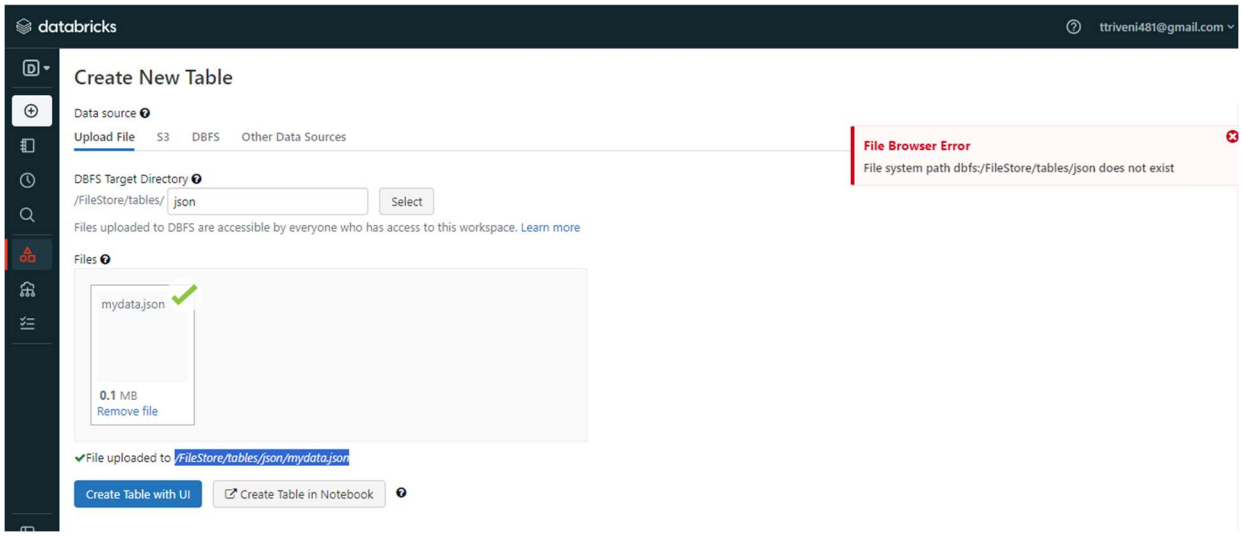
Download the mydata.json from google cloud bucket and stored in the local file system.



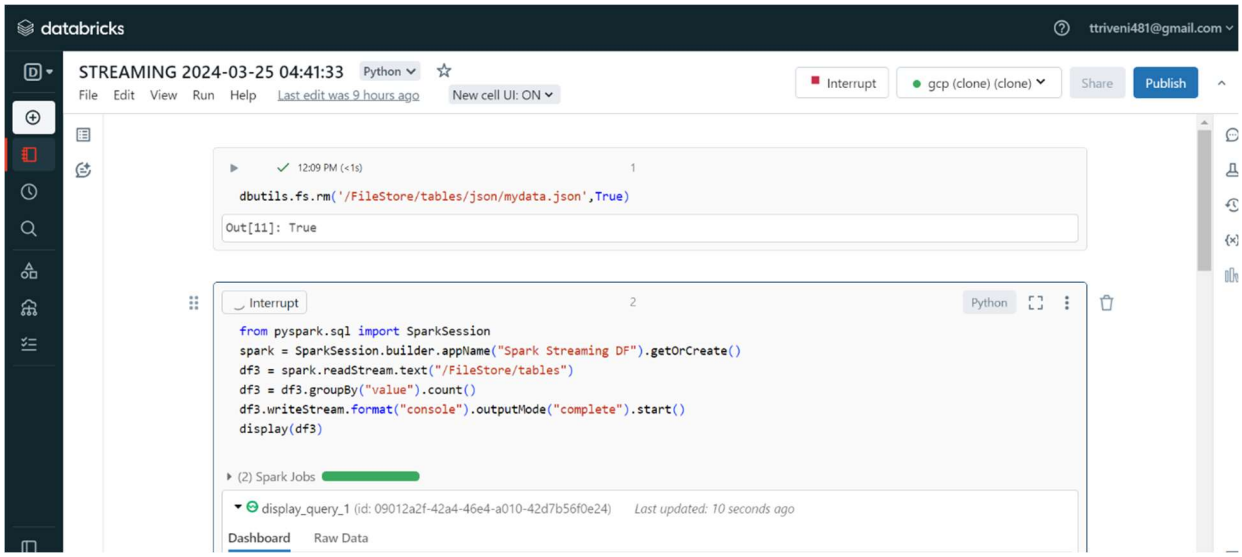
Step 23:

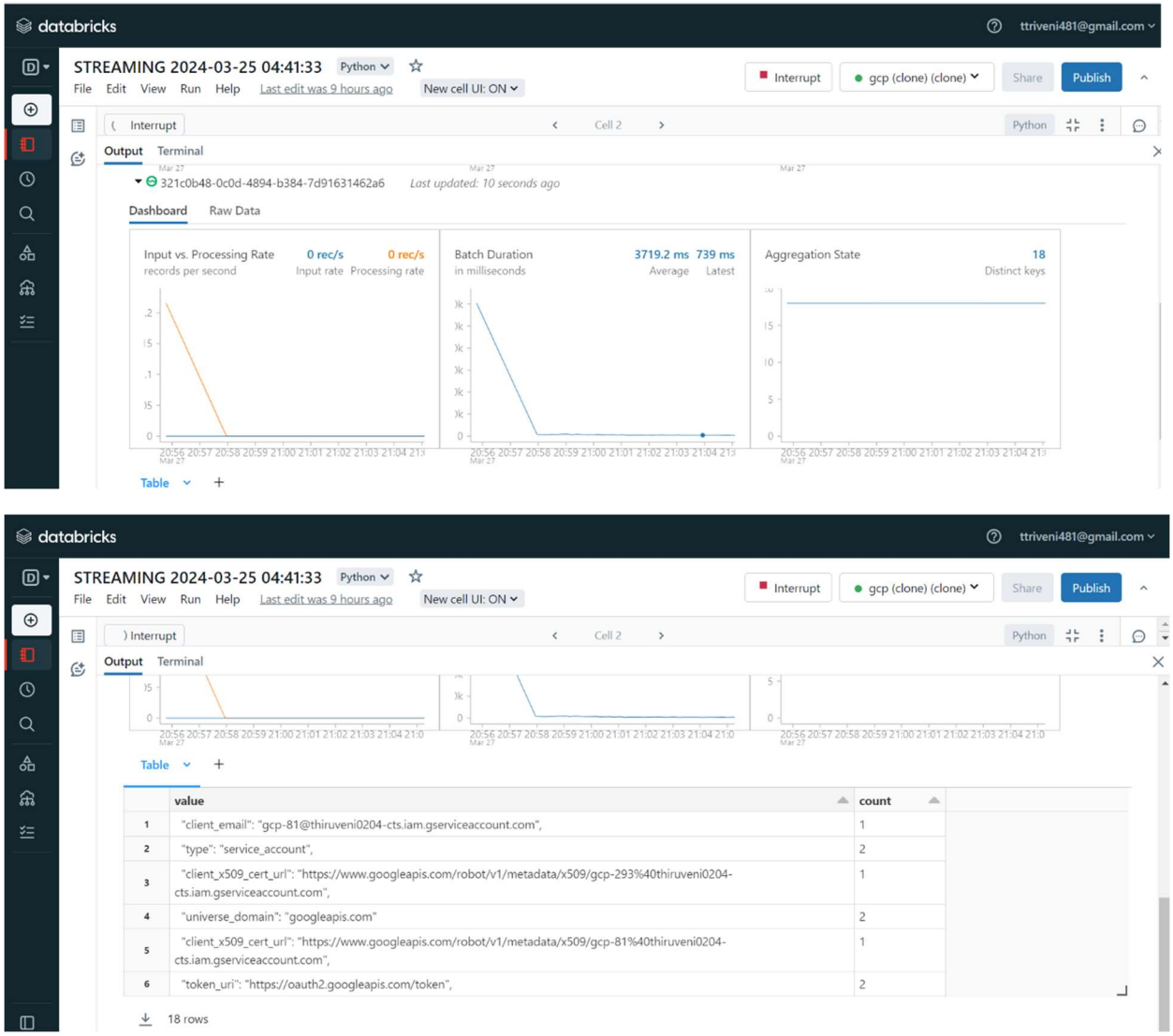
STREAMING:

Upload the downloaded json file into the Databricks table and that was stored in the dbfs.



Streaming the json file using spark session.





CONCLUSION:

In conclusion, streaming serialization in Apache Spark on Databricks offers a powerful solution for real-time data processing and analysis. By leveraging Spark's streaming capabilities and Databricks' cloud-based platform, organizations can efficiently ingest, process, and analyze data streams in a scalable and fault-tolerant manner. Streaming serialization enables the seamless conversion of streaming data into a structured format, facilitating easy integration with Spark's DataFrame and Dataset APIs for further analysis and manipulation. This process ensures that data is efficiently serialized and deserialized during streaming operations, optimizing performance and reducing latency.