**INTERIM PROJECT**

**NAME :** CHAKKA SUDHEER KUMAR

**EMP ID :** 2320418

**COHORT :** CSDAIA24GP003

**DESIGNATION :** GENC INTERN

**TITLE:**

Data Serialization: Serialize Data frame data into various formats (e.g., Avro, ORC, JSON) for storage or exchange.

**OBJECTIVE:**

**The objective of the** project is to transform complex data structures, such as data frames, into a compact and standardized format suitable for storage, transfer, and distribution. By serializing data, we create a byte stream that can be efficiently stored on physical devices or transmitted across networks. The reverse process, called deserialization, reconstructs the original data structures from the serialized byte stream.

**INTRODUCTION:**

**Data serialization**:

It is the process of converting structured data (like arrays, tables or trees) into a format that allows sharing or storage while preserving its original structure. It serves two primary purposes:

1. **Purpose**: Serialization enables data persistence, sharing, and communication between different systems. It ensures that data can be reliably reconstructed even when hardware architecture, operating systems, or addressing mechanisms vary.
2. **Process**:
   1. For simple data (such as numbers or strings), serialization is straightforward.
   2. For nested data structures and object references, serialization becomes more complex. Objects are collapsed into bytes, and information (like traversal order) is included for reconstruction.
   3. Referenced objects are tracked and serialized to avoid redundancy.
   4. All nested objects must be serializable.
3. **Sharing and Storage**: Data serialization enables data to be exchanged across systems, platforms, or programming languages. It allows data to be stored in a form that can be easily recovered later.
4. [**Size Optimization**: In some cases, data serialization aims to minimize the data’s size, which reduces disk space or bandwidth requirements](https://docs.python-guide.org/scenarios/serialization/).
5. **Commonly Used Formats**:
   1. **XML**: A human-readable format based on tags and attributes.
   2. **JSON (JavaScript Object Notation)**: Lightweight, widely used for web APIs and configuration files.

**Different Storage Formats**

1. **Avro**:
   * **Purpose**: Avro is a compact, efficient, and schema-based binary serialization format.
   * **Features**:
     + Schema evolution support (backward and forward compatibility).
     + Efficient for both storage and data exchange.
     + Supports complex data types (arrays, maps, records).
     + Used in Hadoop ecosystems (e.g., Apache Kafka, Apache Spark).
2. **ORC (Optimized Row Columnar)**:
   * **Purpose**: ORC is designed for efficient columnar storage in Hadoop environments.
   * **Features**:
     + Column-oriented storage reduces I/O and improves query performance.
     + Compression and predicate pushdown.
     + Supports complex types.
     + Used in Hive and Impala.
3. **Parquet**:
   * **Purpose**: Parquet is another columnar storage format, widely used in big data ecosystems.
   * **Features**:
     + Columnar layout optimizes query performance.
     + Compression (Snappy, Gzip, etc.).
     + Schema evolution.
     + Suitable for data lakes and data warehouses.
4. **JSON (JavaScript Object Notation)**:
   * **Purpose**: JSON is a human-readable format for data interchange.
   * **Features**:
     + Lightweight and easy to understand.
     + Commonly used in web APIs and configuration files.
     + Not as space efficient as binary formats.
5. **CSV** **(Comma-Separated Values):**

* **Purpose**:
  + A **CSV file** (Comma-separated values) is a simple yet powerful file format used to store and exchange data.
  + **Data Exchange**: CSV files are commonly used for data exchange between different software, databases, and spreadsheet programs.
  + **Lightweight and Efficient**: The lightweight nature of CSV makes it efficient for handling large datasets.
  + **Human and Machine Readable**: Its straightforward structure facilitates easy parsing by both humans and machines.
* **Features:**
  + **Delimiters**: Values within a row are separated using **commas** (though other delimiters like tabs or semicolons can be used).
  + **Quotes**: Quotes allow storing any type of data (including special characters) in a CSV file.
  + **Header Rows**: Header rows help identify column names in the file.
  + **Spreadsheets**: CSV files store spreadsheet data.
  + **Databases**: They are used to import/export data from databases.
  + **Web Applications**: CSV files facilitate data exchange between web applications.
  + **Data Science**: Often used for data storage in data science and machine learning projects.

**Data Deserialization:**

Deserialization refers to the process of converting serialized data (such as Avro, ORC, or Parquet files) back into a structured format, specifically a **Dataframe**.

deserialization involves reading data from files or other sources and converting it into a structured format.

Let’s explore the deserialization process for some common formats:

1. **Avro Deserialization**:
   * **Avro** is a compact binary format that supports schema evolution.
   * To deserialize Avro data into a DataFrame in PySpark, you can use the spark.read.format("avro").load("path/to/avro/files") method.
   * Avro files include schema information, making it easy to read data with varying structures.
2. **ORC Deserialization**:
   * **ORC (Optimized Row Columnar)** is a columnar storage format.
   * To deserialize ORC data, use spark.read.format("orc").load("path/to/orc/files").
   * ORC provides efficient compression and predicate pushdown capabilities.
3. **Parquet Deserialization**:
   * **Parquet** is another columnar storage format that works well with PySpark.
   * Deserialize Parquet data using spark.read.parquet("path/to/parquet/files").
   * Parquet files are highly optimized for analytics workloads.
4. **JSON Deserialization**:
   * While not a binary format, **JSON** is widely used for data exchange.
   * To deserialize JSON data into a DataFrame, use spark.read.json("path/to/json/files").
   * JSON files are human-readable but less space-efficient compared to binary formats.

**Apache Spark:**

A logo on a black background

Description automatically generated

Apache spark is an open-source distributed computing system designed for large-scale data processing. It provides a unified analytics engine with support for various data processing tasks, including batch processing, stream processing, machine learning and graph processing.

**Databricks:**



Databricks is a unified analytics platform designed to simplify the process of building and deploying big data and Al solutions. It is built on Apache Spark, offering a collaborative environment for data engineers, data scientists, and machine learning practitioners to work seamlessly. Databricks provides managed Spark clusters, interactive notebooks, and a suite of tools for data engineering, data science, and machine learning

**IMPLEMENTATION:**

Data Collection:

The first step in serializing the data is to collect the data of table in one of the formats. In this case, the data is generated randomly using python faker code instead doing manually.

**Cell 1-2:** Install faker module for data set creation and also include all pyspark modules & import all functions.

A screenshot of a computer

Description automatically generated

**Cell 3:** faker code to create a employee table giving required columns and convert it into data frame and save it as a csv file.

**A screenshot of a computer

Description automatically generated**

**Cell 4-7:** Using the above data frame, save the table in different formats like avro, orc, parquet, json.

**Cell 8:** read the json file to data frame for further transformations which is called as data deserialization.

**A screenshot of a computer

Description automatically generated**

**Employee table is saved as different storage formats as shown below:**

A screenshot of a computer

Description automatically generated

**Cell 9:** after reading json file, cleansing has to done like making all column values to lower case.

A screenshot of a computer

Description automatically generated

**Cell 10:** Also type cast some columns such as “age, income, premium\_amount”

to integer datatype.

**Cell 11:** write function to categorize the employee data based on their age as follows:

A screenshot of a computer

Description automatically generated

**Cell 12:** using udf keyword (user defined functions), include column ‘Age\_Group’

as follows

A screenshot of a computer

Description automatically generated

**Cell 13**: add extra column “High\_Income” based on column “Income” whose income greater than 80000 will assign ‘yes’ or else ‘no’

**Cell 14:** combine some existing columns like ‘state’ and ‘language’ to a new column ‘state\_language’ using keyword ‘concat\_ws’. It is used to concat two values with a seperator.

A screenshot of a computer

Description automatically generated

**Cell 15:** delete the unnecessary columns using keyword ‘drop’

A screenshot of a computer

Description automatically generated

**Cell 16:** order the data with respective to higher income employees and their premium amounts in ascending order using keyword ‘order by’

A screenshot of a computer

Description automatically generated

**Cell 17-18:** store the employee data after the above transformations in a json file

and read that json file for verification.

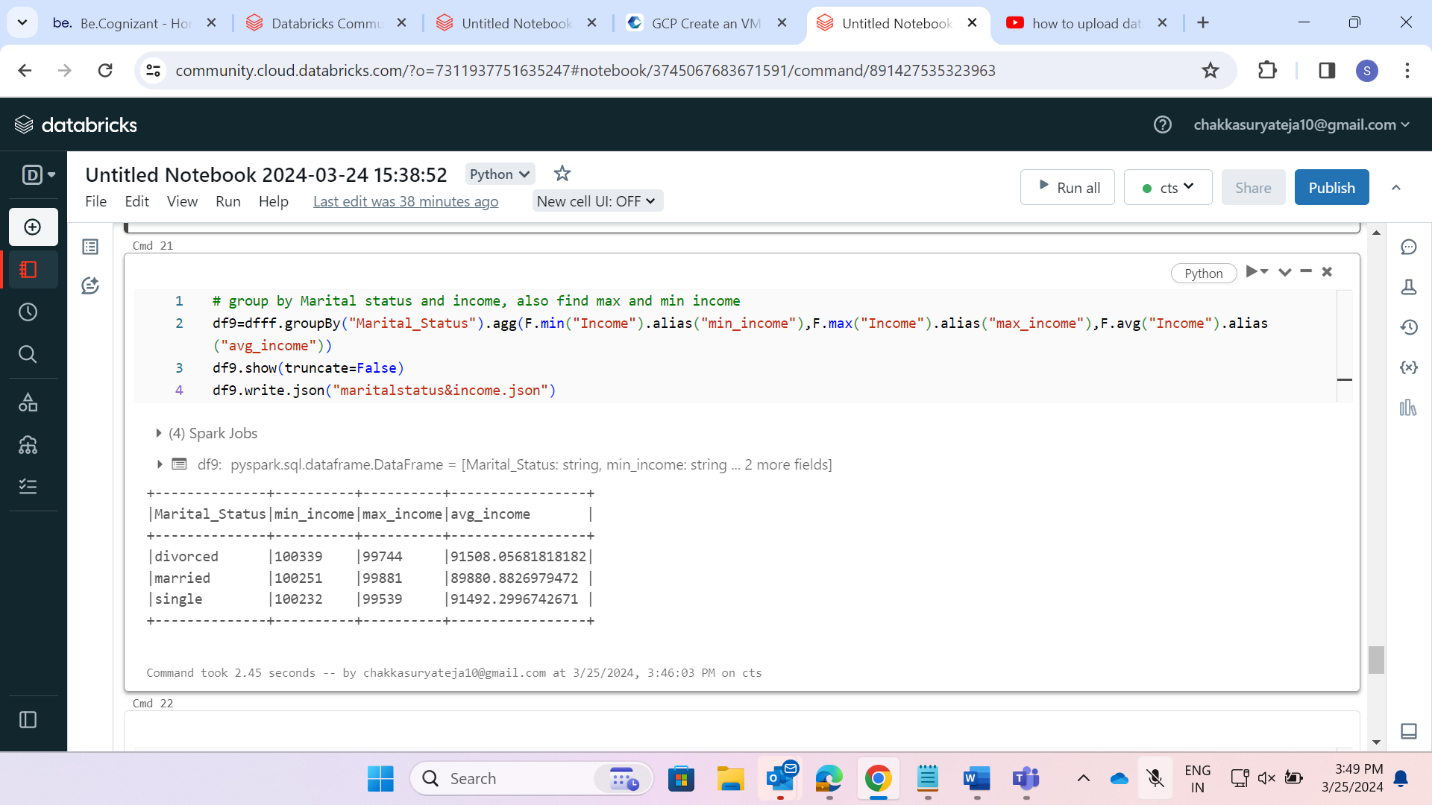


**Cell 19:** filter the employee data whose education is ‘bachelor’ and occupation is ‘teacher’ as follows:

A screenshot of a computer

Description automatically generated

**Cell 20:** group the employee data based on column ‘marital\_status’ and display average income using ‘income’ column and save the data in a json file as follows:



**Cell 21:** group the employee data based on column ‘Gender’ and display count of male and female employee using keyword ‘count’ follows:

A screenshot of a computer

Description automatically generated

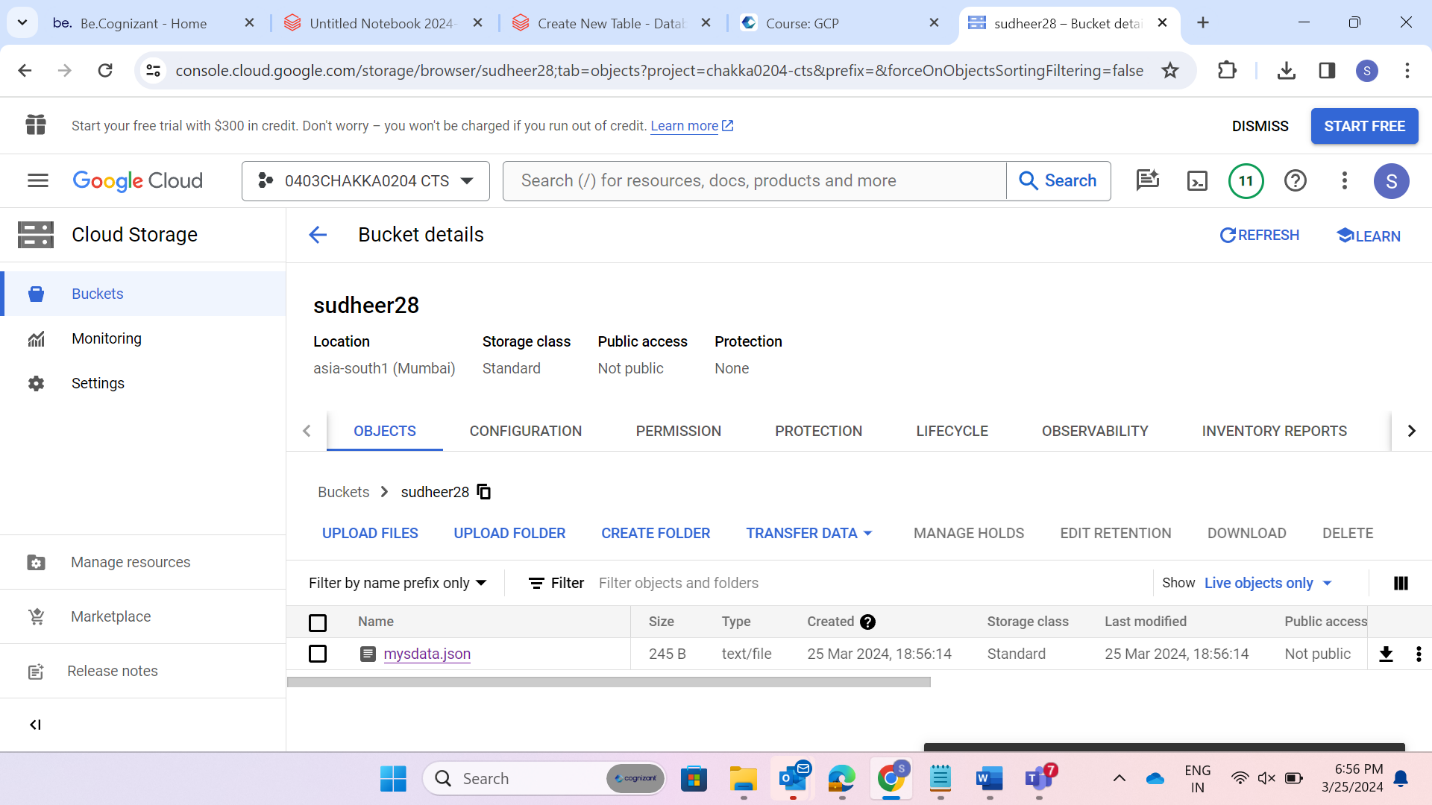
**Cell 22:** group the employee data based on column ‘State’ and display count of employees residing in each state using keyword ‘count’ and save the data in a json file as follows:

**A screenshot of a computer

Description automatically generated**

**Importing JSON Files to Google Cloud Storage:**

* Create a Bucket in Google Cloud Storage and Service Account in IAM admin service section.
* Change the permissions in the created bucket ‘sudheer28’ and download Json file.

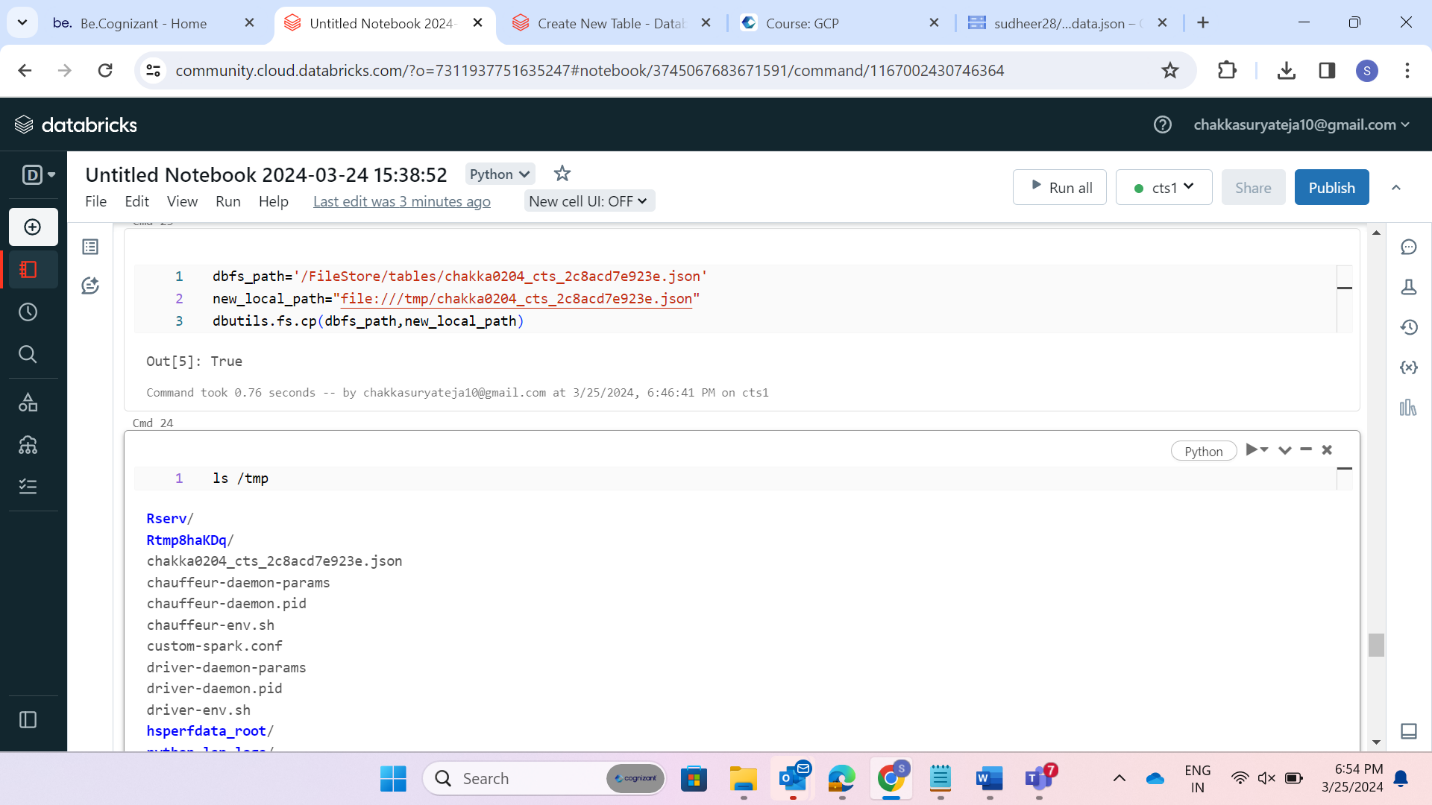


* Now upload that Json file to data bricks and copy the path of the Json file in data bricks file system.

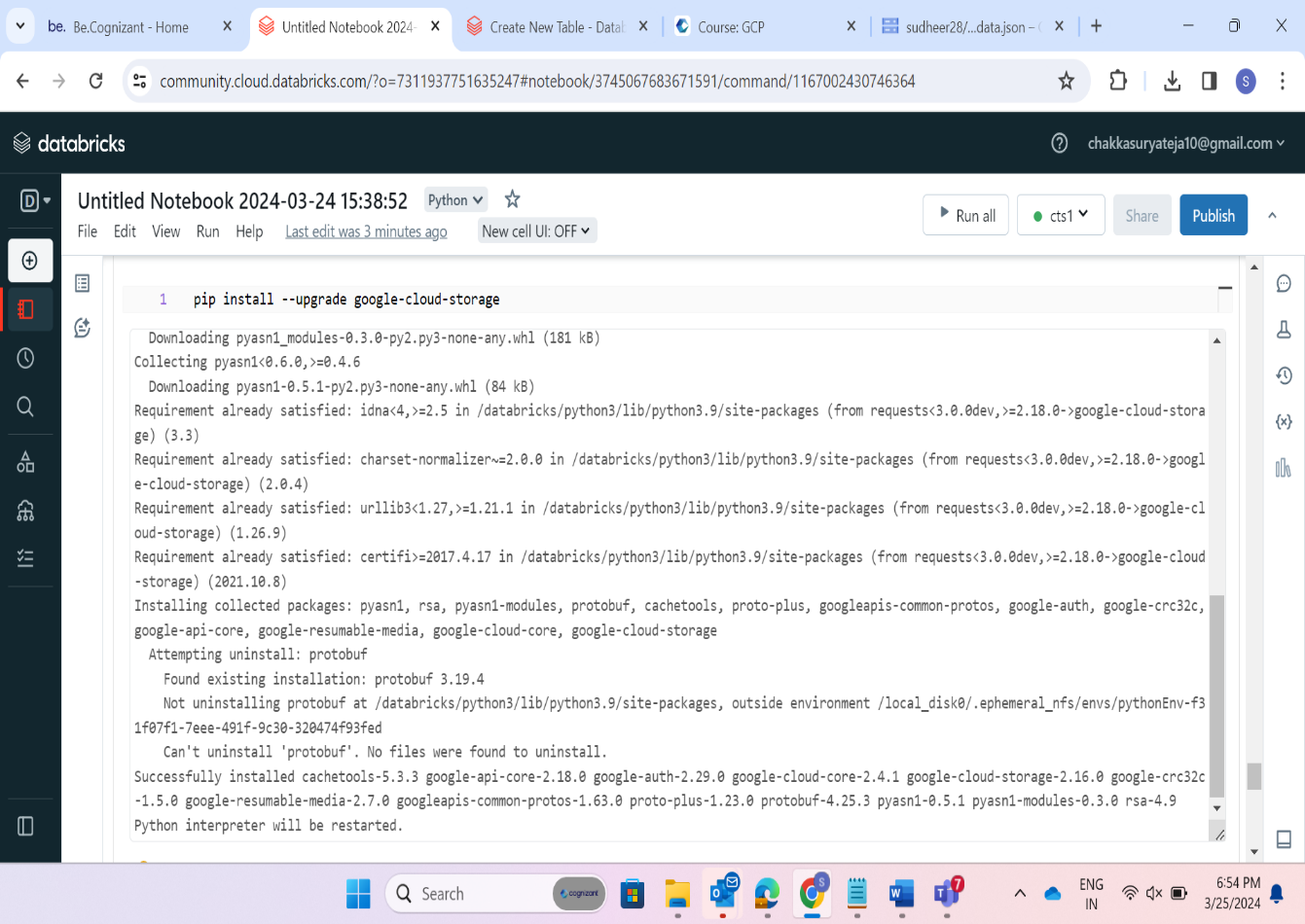
A screenshot of a computer

Description automatically generated

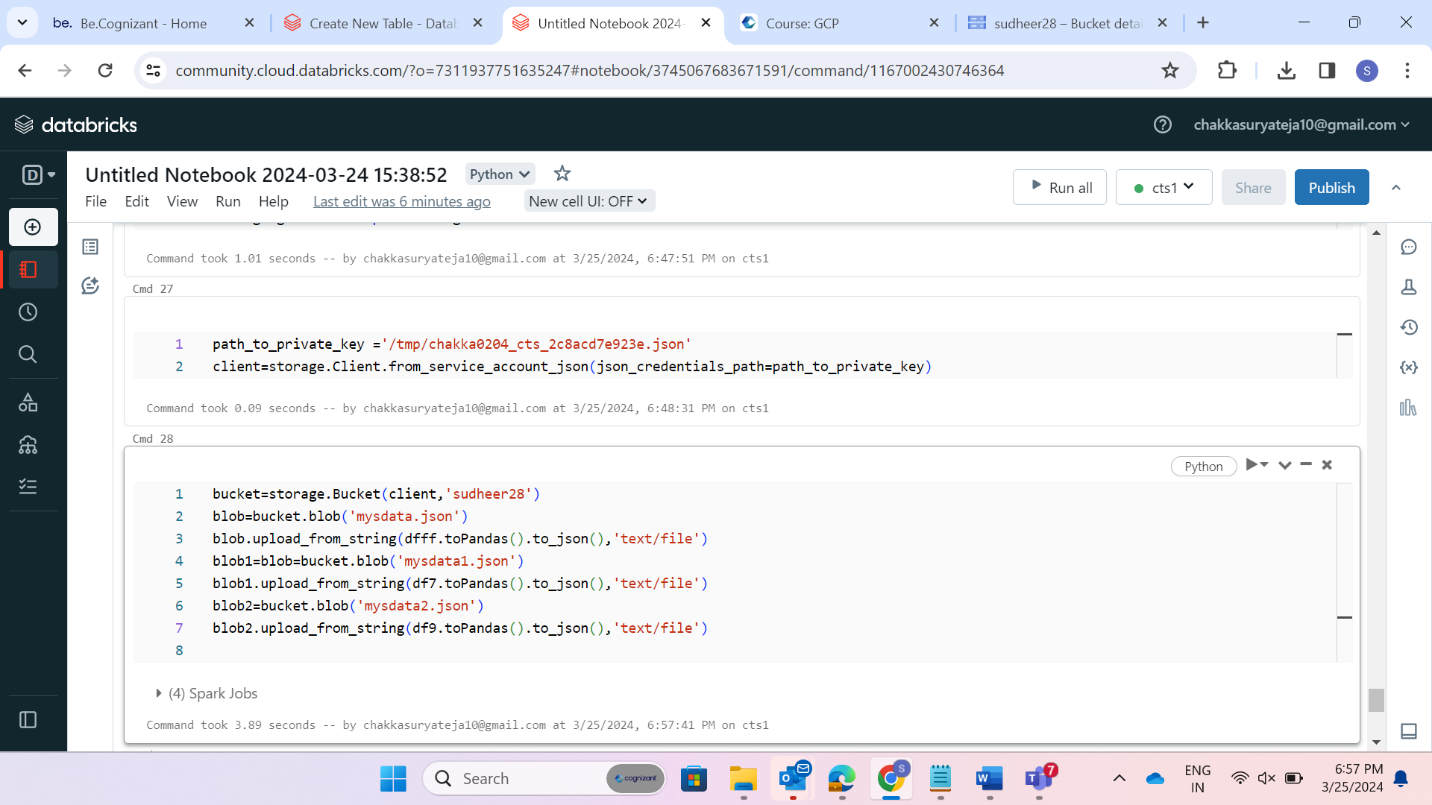
* Now include Json file path & local path and also include some other important commands.



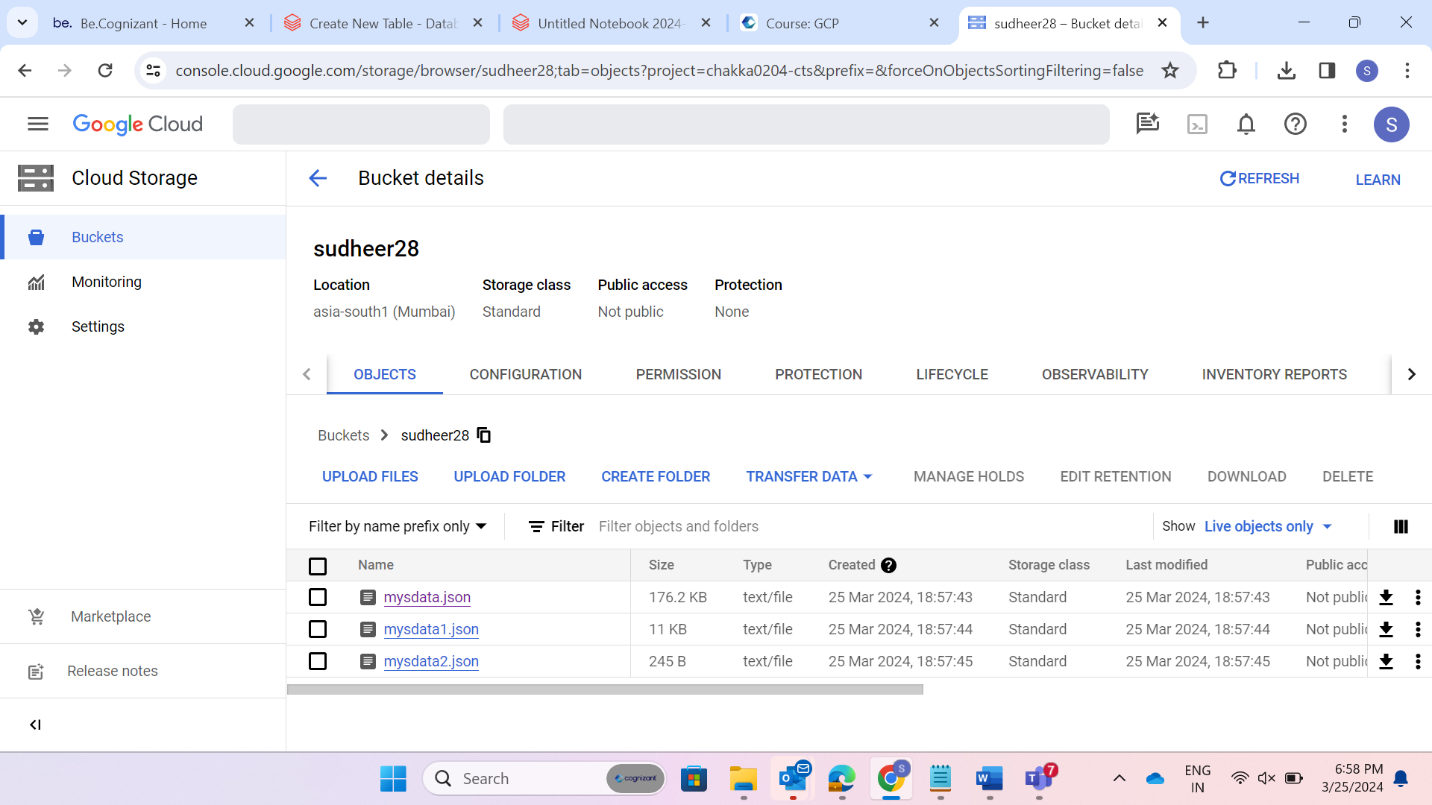
* Include google cloud storage module to data bricks by running below command.



* Write the commands below to import data frame to cloud storage as follows:



* Now we can see imported Json files to google cloud bucket as shown below:



## **Real-Time Applications:**

1. **Apache Kafka**:
   * In Kafka, data serialization is crucial for seamless data transfer and compatibility.
   * **Avro** provides schema-based serialization, enabling schema evolution and language independence.
   * **JSON** offers a widely supported and human-readable format for serialization.
2. **Big Data Processing**:
   * In distributed computing frameworks like **Hadoop** and **Spark**, data serialization is essential for efficient data exchange between nodes.
   * Formats like **Avro** and **ORC** (Optimized Row Columnar) are commonly used for storing and processing large datasets.
3. **Microservices Communication**:
   * Microservices architectures rely on efficient communication between services.
   * Lightweight serialization formats like **JSON** or **MessagePack** are commonly used for inter-service communication.

## **Advantages:**

1. **Compact Representation**:
   * Serialized data is typically more compact than its original form.
   * This reduces storage requirements and improves data transfer efficiency.
2. **Language Independence**:
   * Serialized data can be deserialized in any language that supports the chosen format.
   * This enables interoperability between systems written in different languages.
3. **Schema Evolution**:
   * Formats like **Avro** allow for schema evolution without breaking compatibility.
   * New fields can be added to the schema, and existing data can still be read.

## **Conclusion**:

Data serialization is a fundamental concept in modern software development. By choosing the right serialization format based on factors like data complexity, readability, speed, and storage space, developers can ensure efficient data exchange and seamless integration across diverse systems. Whether you’re working with PySpark, Kafka, or any other technology stack, understanding data serialization is essential for building robust and scalable applications.