

GOOGLE CLOUD PLATFORM FIRESTORE FINAL PROJECT



Google Cloud

Building a Scalable NoSQL Database with Firestore

**Nandan Prabhu
CSDAIA24GP003
2320667**

Introduction

GCP, or Google Cloud Platform, is a suite of cloud computing services offered by Google. It provides a variety of tools and resources that allow businesses and organizations to run their applications and data centres on Google's infrastructure.

Here's a quick breakdown of GCP:

- **Services:** It offers a range of services including compute, storage, data analytics, and machine learning. Think of storage for housing your data, compute for running applications, analytics for crunching numbers, and machine learning for building intelligent models.
- **Scalability:** GCP is known for its scalability, meaning you can easily adjust your resource usage up or down based on your needs.
- **Security:** Security is a major focus of GCP, with Google using the same infrastructure for GCP that it uses for its own products, like Gmail and Search.
- **Benefits:** Businesses use GCP for a variety of reasons, including cost savings, increased agility, and access to advanced technologies like AI and machine learning.

Core GCP Services:

- **Compute Engine:** This service provides virtual machines (VMs) that you can use to run your workloads on Google's infrastructure. VMs offer a familiar computing environment for developers.
- **Cloud Functions:** This serverless offering lets you run code without having to manage servers yourself. It's ideal for event-driven applications or small tasks.

- **Kubernetes Engine:** This managed Kubernetes service allows you to deploy and manage containerized applications in Google Cloud. Kubernetes is a popular container orchestration platform.
- **Cloud Storage:** This offers scalable and durable storage for a variety of data types, from archives to frequently accessed objects.
- **BigQuery:** A data warehouse service for large datasets. BigQuery allows you to analyse massive datasets quickly and efficiently.

Advantages of GCP:

- **Cost-Effectiveness:** GCP offers a pay-as-you-go pricing model, so you only pay for the resources you use.
- **Scalability:** Easily scale resources up or down to meet your needs.
- **Security:** GCP offers robust security features to protect your data and applications.
- **Reliability:** Google's global infrastructure ensures high availability and reliability for your applications.
- **Open Source Friendly:** GCP embraces open-source technologies and offers integration with many popular open-source tools.

FIRESTORE

Firestore, a key component of Firebase and Google Cloud Platform (GCP), is a flexible and scalable NoSQL database designed for mobile, web, and server development. Here's a comprehensive breakdown of Firestore:

What it is:

- **NoSQL Document Database:** Firestore stores data in JSON-like documents, offering a flexible schema compared to traditional relational databases.
- **Cloud-Hosted:** You don't need to manage or maintain servers yourself. Google takes care of the infrastructure.
- **Offline Support:** Firestore caches data locally on devices, enabling your app to function even without an internet connection.

Key Features:

- **Rich Querying:** Firestore allows complex queries to retrieve specific data efficiently. You can filter, sort, and perform aggregations on your data.
- **ACID Transactions:** Firestore ensures Atomicity, Consistency, Isolation, and Durability (ACID) for data operations, guaranteeing data integrity.
- **Offline Persistence:** As mentioned earlier, Firestore caches data locally, allowing users to interact with data even when offline. Changes are synced back to the cloud when the device regains connectivity.
- **Real-time Updates:** Firestore keeps data synchronized across connected devices in real-time using listeners. This is ideal for collaborative apps or applications where data updates need to be reflected immediately for all users.
- **Scalability:** Firestore scales seamlessly to accommodate growing data volumes and user bases. You won't need to worry about managing server capacity.
- **Security:** Firestore integrates with Google Cloud's security tools, allowing you to implement granular access controls to protect your data.

- **Integration with other GCP Services:** Firestore integrates well with other GCP services like Cloud Functions for building serverless triggers based on database events.

Use Cases:

- **Mobile and Web Apps:** Firestore's offline capabilities and real-time updates make it a popular choice for building dynamic mobile and web applications.
- **Social Applications:** Features like real-time data updates are perfect for social apps where users need to see changes instantaneously.
- **E-commerce Applications:** Firestore can efficiently manage product catalogs, user data, and shopping carts.
- **Real-time Collaboration:** Firestore is well-suited for collaborative applications where multiple users need to work with the same data simultaneously.

Benefits of using Firestore:

- **Flexibility:** The document-based schema allows you to store data in a way that aligns with your application's needs.
- **Scalability:** Easily handle growing data volumes without worrying about infrastructure limitations.
- **Offline Support:** Provides a seamless user experience even in situations with unreliable internet connectivity.
- **Real-time Updates:** Keeps all connected devices in sync, offering an up-to-date view of the data.
- **Security:** Leverage Google Cloud's robust security features to protect your data.

TASKS:

- 1.Create a firestore database
- 2.Create one collection
- 3.Add few documents
- 4.Create python script which will read data from a file to firestore.

OBJECTIVE:

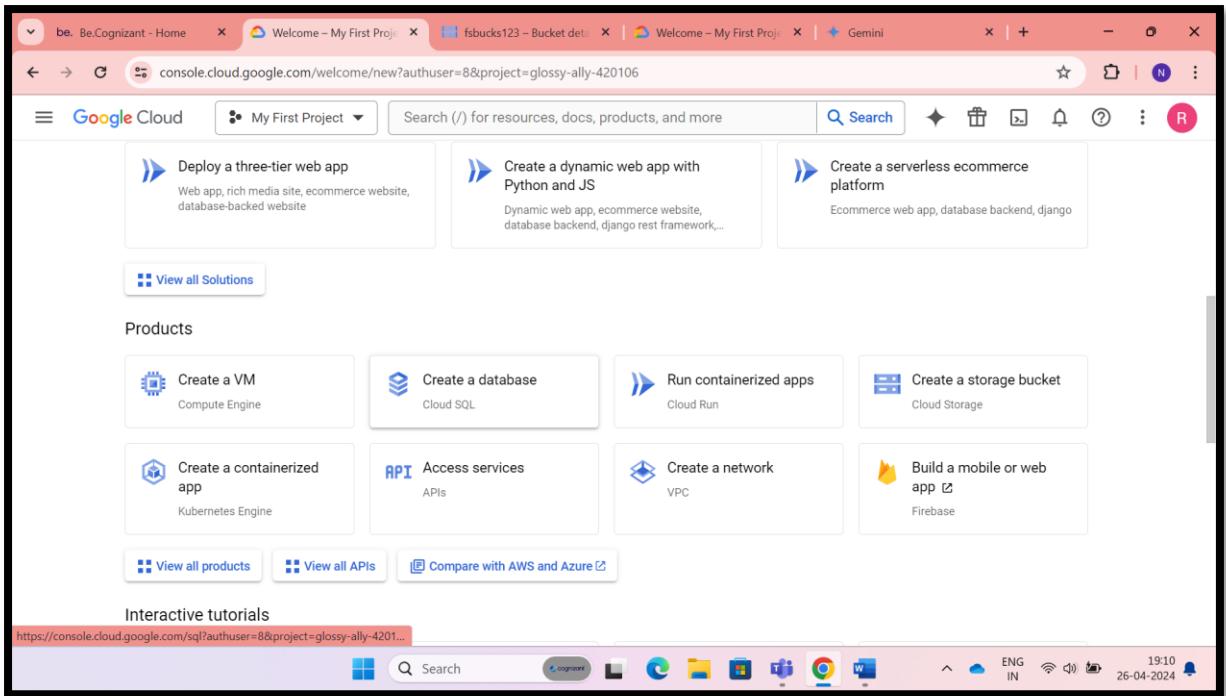
To set up a Firestore database, create a collection, add documents to the collection, and develop a Python script to import data from a file into Firestore.

WAYS TO INTERACT WITH GCP:

Google Cloud gives you three basic ways to interact with the services and resources.

1.Google cloud console (GUI):

- The **Google Cloud Console** is a web-based interface that allows you to manage GCP resources visually.
- You can create, configure, and monitor services, virtual machines, databases, and more.
- It's beginner-friendly and provides an intuitive way to interact with GCP.
- When you use the Google Cloud console, you either create a new project or choose an existing project, and then use the resources that you create in the context of that project.



2. Command-Line Interface (CLI):

- If you prefer to work at the command line, you can perform most Google Cloud tasks by using the google cloud cli. The gcloud CLI lets you manage development workflow and Google Cloud resources in a terminal window.
- The **gcloud CLI tool**, part of the **Google Cloud SDK**, lets you interact with GCP services via the command line.
- Use it for tasks like creating instances, managing storage, configuring networking, and deploying applications.
- The CLI is powerful for scripting and automation.

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to glossy-ally-420106.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
ramanagcp29@cloudshell:~ (glossy-ally-420106)$ ls
delete.sh          file.json    firestore-read.py  glossy-ally-420106-49fb51f6c1.json   README-cloudshell.txt  students.json
ecommerce_data.csv  firestore  'from google.py'  glossy-ally-420106-8db53095cd3e.json  student_data.json
ramanagcp29@cloudshell:~ (glossy-ally-420106)$ firestore --help
-bash: firestore: command not found
ramanagcp29@cloudshell:~ (glossy-ally-420106)$
```

3. Python Client Libraries:

- GCP offers **Python client libraries** for various services.
- You can use these libraries in your Python code to interact with GCP programmatically.
- For example, you can create virtual machines, manage storage buckets, and access APIs directly from your Python scripts.

```
import.py      create.py  super_db.json  sorted.json
firestore > create.py > [0]
1  from google.cloud import firestore
2
3  db = firestore.Client(project="glossy-ally-420106",database="mydatabase")
4
5  data = [{"Emp_ID": "EMP001","Name": " Sam Curran","position": "Cashier", "contact":{ "Email": "sam.curran@super.com", "phone": "9876543210"}, "address": "123 Main St, Anytown, USA"}, {"Emp_ID": "EMP002","Name": " MS Dhoni","position": "Store Manager", "contact":{ "Email": "msdhoni@super.com", "phone": "9876543211"}, "address": "456 Market St, Anytown, USA"}]
6
7  for i in data:
8      doc = db.collection("customers").document("136GKFFRu68BvTr0VeZW")
9      doc.set(i,merge=True)
10
11      print("Added Successfully")
```

IMPLEMENTATION

- 1. Basic Firestore Creation and Operations**
- 2. Firestore Integration with Other GCP Services**

IMPLEMENTATION :1

- **Basic Firestore Creation and Operations**

Task 1: Create a Firestore Database

There are two modes in firestore while creating database. They are:

- Native mode
- Datastore mode

The main difference is:

1. Firestore in Native mode:

- Next major version of Datastore.
- Combines Datastore and Firebase Realtime Database features.
- Offers real-time updates, a new data model, and mobile/web client libraries.

2. Firestore in Datastore mode:

- Uses Datastore system behaviour but accesses Firestore's storage layer.
- Removes Datastore limitations (e.g., eventual consistency, transaction limits).
- Disables Firestore features not compatible with Datastore.

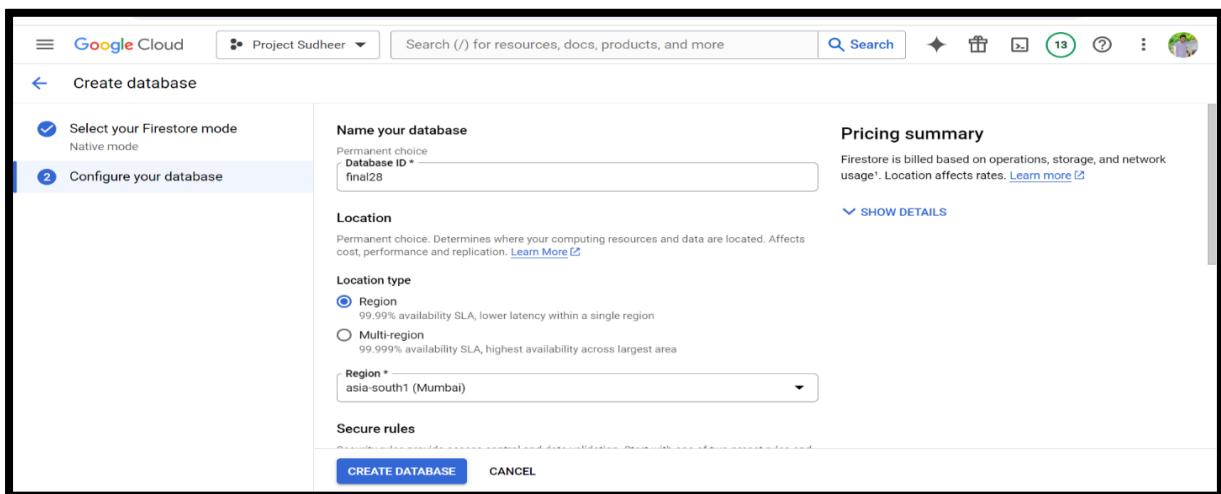
3. Use Cases:

- **Native mode:** Ideal for real-time apps, scalability, and strong consistency.

- o **Datastore mode:** Suitable for flexible data modelling and hierarchical data.

GUI Interactive way:

- Select Native mode and create database



CLI way:

- In command line interface way, it is limited to do certain database operations in firestore.

```
Available commands for gcloud firestore databases:
create
Create a Google Cloud Firestore database via Firestore API.

delete
Delete a Google Cloud Firestore database.

describe
Describes information about a Cloud Firestore database.

list
Lists all Firestore databases under the project.

update
Update the database configuration of a Cloud Firestore database.
```

Command to create database in firestore is:

```
gcloud firestore databases create --database=DATABASE_ID --
location=LOCATION --type=DATABASE_TYPE \
[--delete-protection]
```

The screenshot shows the Google Cloud Platform interface with the Firestore section selected. The database list table includes columns for Database ID, Mode, Location, and Creation time (UTC+5:30). Four databases are listed: (default), ecommercecloud, final28, and final30, all in Native mode and located in asia-south1. Below the table is a terminal window titled '(project-sudheer-a5abf)' showing the command to create a new database:

```
Update the database configuration of a Cloud Firestore database.

For detailed information on this command and its flags, run:
gcloud firestore databases --help
chakkandu@chakkandu-OptiPlex-5090: ~ (project-sudheer-a5abf)$ gcloud firestore databases create --location=asia-south1 --database=final30
metadata:
  '@type': type.googleapis.com/google.firebaseio.admin.v1.CreateDatabaseMetadata
name: projects/project-sudheer-a5abf/databases/final30/operations/d2xJh7zQc4tLSRZYajQPhAgMWh0dW9zLWppc2ELIgIQAcWx39AQBrGPzPMIDAvGg
response:
  '@type': type.googleapis.com/google.firebaseio.admin.v1.Database
  appEngineIntegrationMode: DISABLED
  concurrencyMode: PESSIMISTIC
  createTime: '2024-04-20T15:59:47.413954Z'
```

Task 2: Creation of collection

- Collection creation is possible only with GUI and Python way, but we can't create collections and documents using CLI.

The screenshot shows the Google Cloud Platform interface with the Firestore Studio section selected. A collection named 'supermarket' is being created. The 'Give the collection an ID' step shows a 'Collection ID' of 'Employees'. The 'Add its first document' step shows a document with fields: Emp_ID (string, value: EMP001), Name (string, value: John Wick), Position (string, value: Store Manager), and Contact (map). The Contact field has two sub-fields: Email (string, value: john.wick@super.com) and Phone (string, value: 9012373456).

Task 3: Add few documents in the collection.

- Add few documents to existing collection and documents has its own unique id.

The screenshot shows the Google Cloud Firestore Studio interface. On the left, there's a sidebar with options like Firestore Studio, Indexes, Import/Export, Disaster Recovery, Time-to-live (TTL), and Security Rules. The main area shows a database named 'supermarket' with a collection 'Employees'. A new document is being created with the ID 'QhMmj6vVzlO6ljf0Rz0'. The document contains fields: Email ('mark.antony@super.com'), Phone ('8912345678'), Emp_ID ('EMP003'), Name ('Mark Antony'), and Position ('Stock Clerk'). A message at the bottom says 'Created document QhMmj6vVzlO6ljf0Rz0'.

Task 4: Create python script which will read data from a file to firestore.

- Python script is executed on the editor provided by GCP with requires authorization.

The screenshot shows the Google Cloud Shell Editor. The left sidebar lists files: import.py, super_db.json, sorted.json, create.py, delete_dummy.py, delete.py, fscommands, gcs_to_firestore.py, import.py (selected), query.py, read.py, sortedjson, super_db.json, update.py, delete.sh, ecommerce_data.csv, file.json, and firestore-read.py. The main area shows the content of 'import.py':

```

import json
from google.cloud import storage
from google.cloud import firestore

# Initialize Firestore client with specific credentials and database
db = firestore.Client(project="glossy-ally-420106", database="mydatabase")

# Initialize Google Cloud Storage client
storage_client = storage.Client()

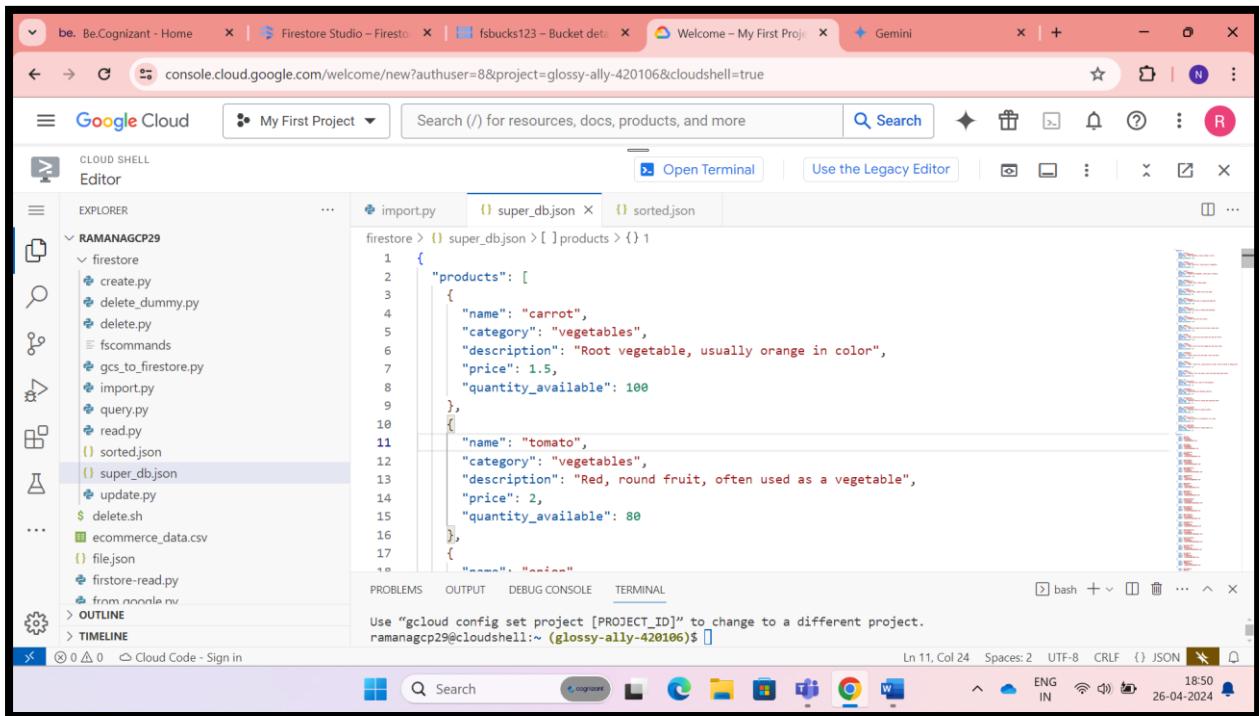
# Define your bucket and JSON file name
bucket_name = "fsbuckets123"
json_file_name = "super_db.json"

# Download the JSON file from the Google Cloud Storage bucket

```

Below the code, the terminal shows the command being run: 'ramanagcp29@cloudshell:~\$ /bin/python /home/ramanagcp29/firestore/import.py'. The output indicates success: 'Data imported Successfully'. The status bar at the bottom shows the terminal is in Python mode.

- This is the JSON file used to write data into FIRESTORE.



```

super_db.json
1  {
2    "products": [
3      {
4        "name": "carrot",
5        "category": "vegetables",
6        "description": "Root vegetable, usually orange in color",
7        "price": 1.5,
8        "quantity_available": 100
9      },
10     {
11       "name": "tomato",
12       "category": "vegetables",
13       "description": "Red, round fruit, often used as a vegetable",
14       "price": 2,
15       "quantity_available": 80
16     },
17     {
18       "name": "onion"
19     }
20   ]
21 }
```

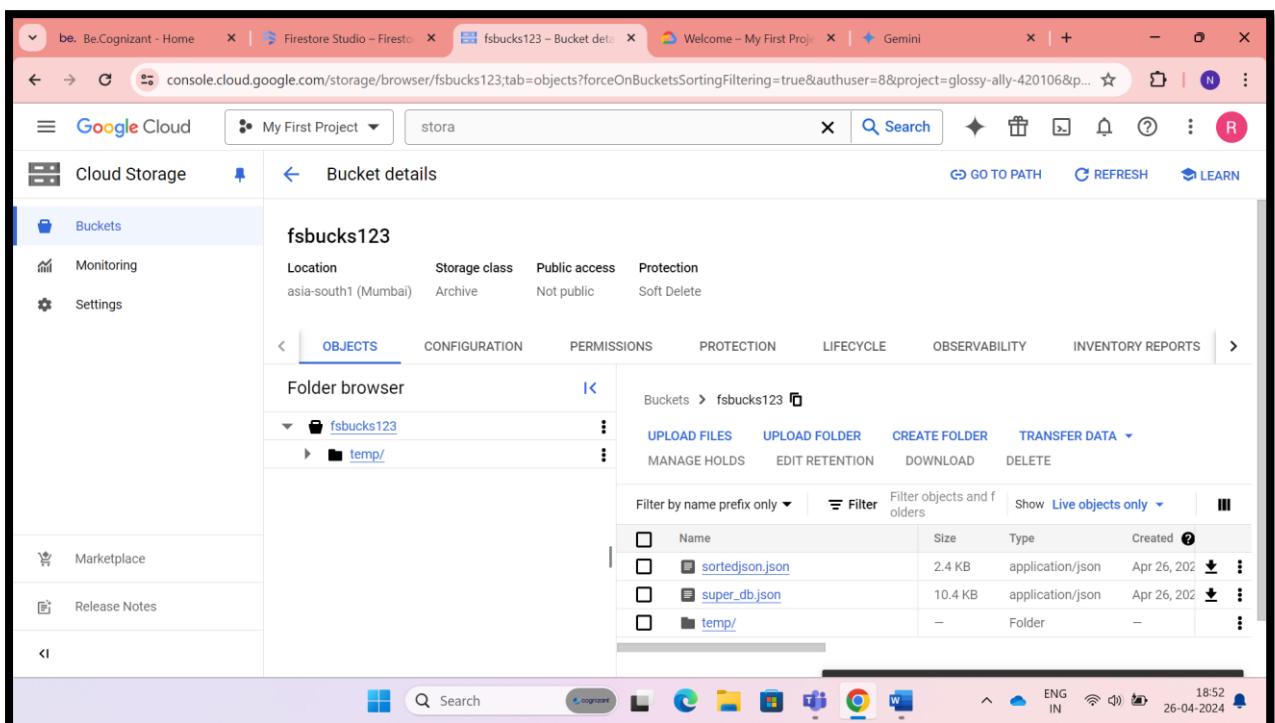
The screenshot shows the Google Cloud Shell Editor interface. The left sidebar displays a file tree for a project named 'RAMANAGCP29' containing various Python scripts and configuration files. The main editor area shows a JSON file named 'super_db.json' with the following content:

```

{
  "products": [
    {
      "name": "carrot",
      "category": "vegetables",
      "description": "Root vegetable, usually orange in color",
      "price": 1.5,
      "quantity_available": 100
    },
    {
      "name": "tomato",
      "category": "vegetables",
      "description": "Red, round fruit, often used as a vegetable",
      "price": 2,
      "quantity_available": 80
    },
    {
      "name": "onion"
    }
  ]
}
```

The bottom status bar indicates the terminal is running on 'ramanagcp29@cloudshell:~ (glossy-ally-420106)\$'.

- I have used Google cloud storage as pipeline for writing data to FIRESTORE.



The screenshot shows the Google Cloud Storage 'Bucket details' page for a bucket named 'fsbucks123'. The left sidebar lists 'Cloud Storage' and 'Buckets'. The main area shows the bucket configuration and an 'OBJECTS' tab with a 'Folder browser' view. The 'OBJECTS' table lists the following files:

Name	Type	Created	Actions
sortedjson.json	application/json	Apr 26, 2024	⋮
super_db.json	application/json	Apr 26, 2024	⋮
temp/	Folder	—	⋮

The bottom status bar indicates the terminal is running on 'ramanagcp29@cloudshell:~ (glossy-ally-420106)\$'.

- After execution the Collections have been created with Documents with all data inside it.

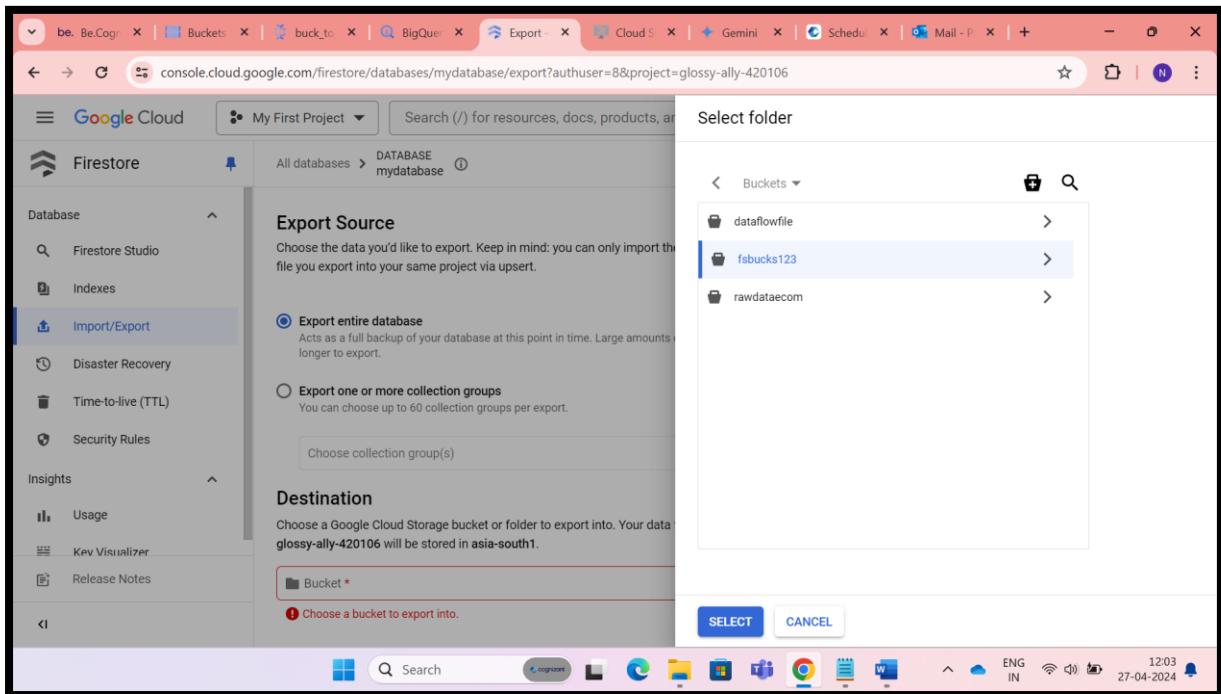
The screenshot shows the Google Cloud Firestore interface. On the left, the sidebar includes 'Firestore Studio', 'Indexes', 'Import/Export', 'Disaster Recovery', 'Time-to-live (TTL)', 'Security Rules', 'Insights' (with 'Usage' and 'Key Visualizer'), and 'Release Notes'. The main area shows a database named 'mydatabase' under 'All databases'. Inside 'mydatabase', there is a collection named 'customers'. Under 'customers', there is a document with the ID '1xB00ooAo4cGfA8FkPGv'. This document contains three sub-collections: 'orders' (with one document '4z7xSaLhJMAcTuQ5sSep'), 'products' (with two documents '6lan1HfVRmx7gHeu2k3W' and '8q824blaKAB2wjY5bOOL'), and a field '1xB00ooAo4cGfA8FkPGv' which contains four key-value pairs: 'city: "Boston"', 'email: "chlolee@example.com"', 'name: "ChloeLee"', and 'phone: "+1122334455"'. The top navigation bar shows multiple tabs like 'Dashboard', 'Mail - Prabh...', 'Welcome - M...', 'Google Cloud', 'fbucks123...', 'Gemini', and others.

Additional Tasks:

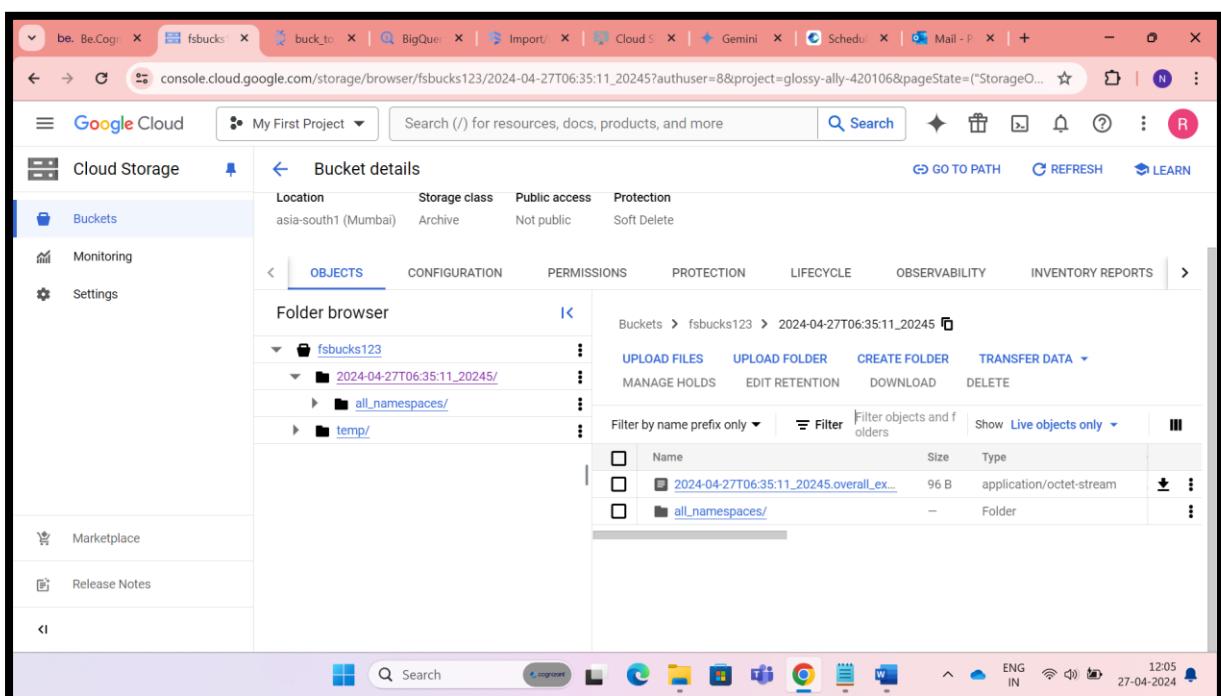
- Import/Export of data from firestore.

The screenshot shows the Google Cloud Firestore interface focusing on the 'Import/Export' section. The sidebar is identical to the previous screenshot. The main area has a heading 'Import/Export' with 'IMPORT' and 'EXPORT' buttons. Below this, it says 'Copy data between Firestore projects and GCS here, or using [Cloud Shell](#). Export regularly, and before major updates, to back up your data and safeguard against error. [Learn more](#)'. It also mentions 'Import/Export jobs run as: service-216877797798@gcp-sa-firebase.iam.gserviceaccount.com'. A 'Filter' section allows filtering by 'Started', 'Type', 'Collection groups', 'Bucket', 'Documents', 'Size', and 'Completed'. A note states 'You haven't moved any data recently.' The top navigation bar is similar to the first screenshot.

- Exporting data from firestore to Google cloud storage.

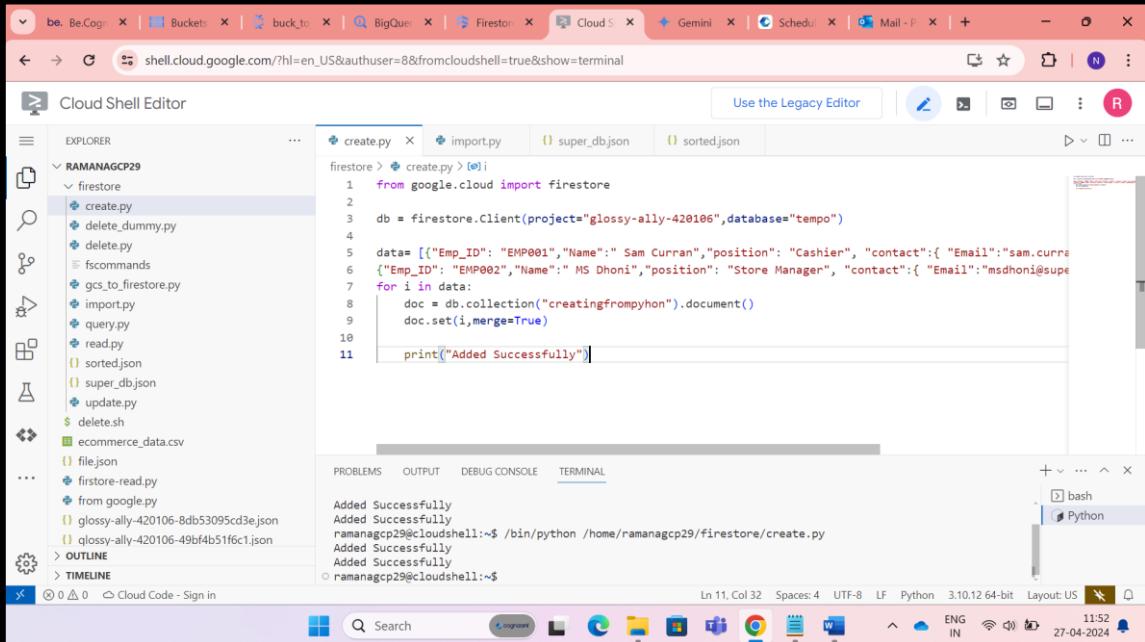


- Successfully data is exported.



➤ Some CRUD operations using python.

➤ Creating collection and adding documents using python.



The screenshot shows the Google Cloud Shell Editor interface. In the left sidebar, there's an 'EXPLORER' section with a tree view of files in the 'RAMANAGCP29' project. The 'firestore' folder contains several Python scripts like 'create.py', 'delete_dummy.py', etc., along with JSON files 'sorted.json' and 'super_db.json'. The main editor area has tabs for 'create.py', 'import.py', 'super_db.json', and 'sorted.json'. The 'create.py' tab contains the following Python code:

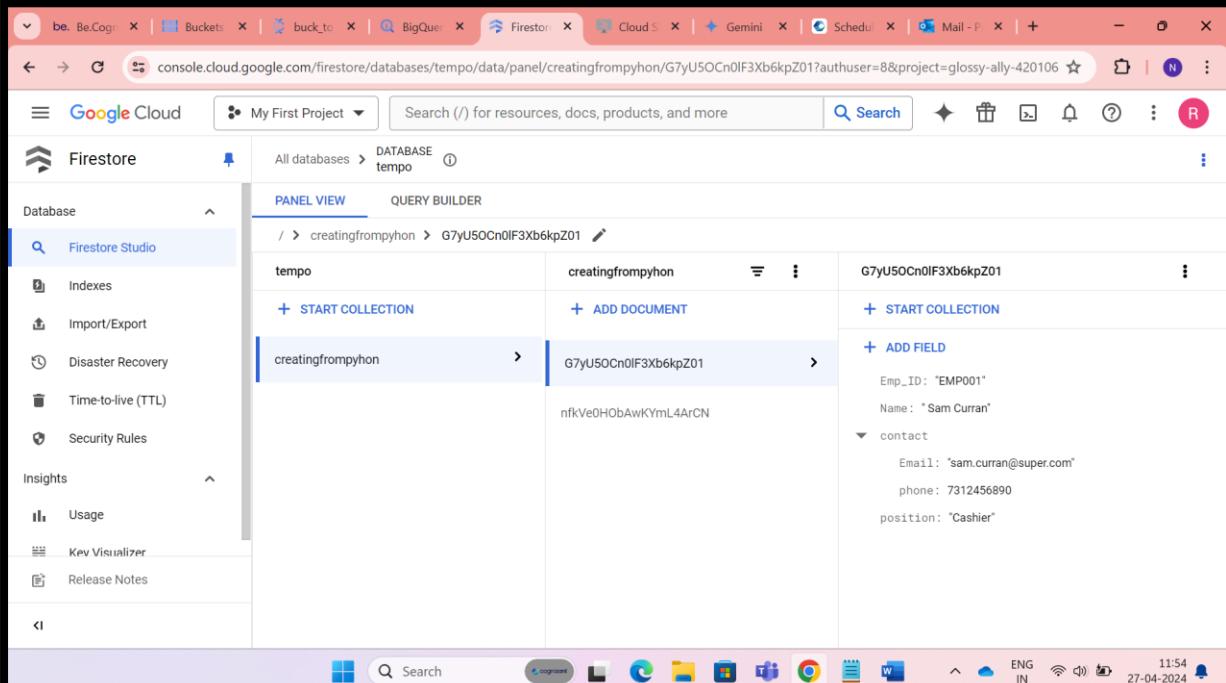
```
from google.cloud import firestore
db = firestore.Client(project="glossy-ally-420106", database="tempo")
data = [{"Emp_ID": "EMP001", "Name": " Sam Curran", "position": "Cashier", "contact": {"Email": "sam.curran@super.com", "Phone": "7312456890"}, "Address": "123 Main St"}, {"Emp_ID": "EMP002", "Name": " MS Dhoni", "position": "Store Manager", "contact": {"Email": "msdhoni@super.com", "Phone": "9876543210"}, "Address": "456 Market St"}]
for i in data:
    doc = db.collection("creatingfrompython").document()
    doc.set(i, merge=True)
print("Added Successfully")
```

Below the code, the terminal window shows the output of running the script:

```
Added Successfully
Added Successfully
Added Successfully
Added Successfully
Added Successfully
ramanagcp29@cloudshell:~$ /bin/python /home/ramanagcp29/firestore/create.py
Added Successfully
Added Successfully
Added Successfully
Added Successfully
Added Successfully
ramanagcp29@cloudshell:~$
```

The status bar at the bottom indicates the session is in bash mode, using Python 3.10.12 64-bit, and the layout is US.

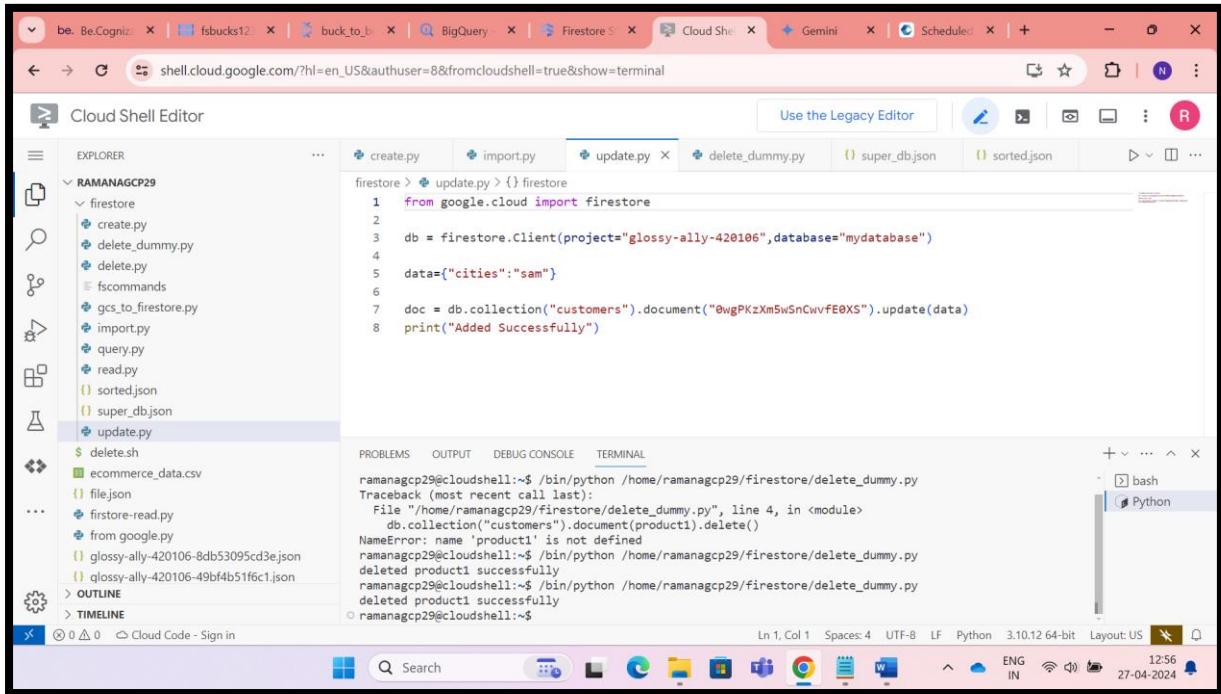
➤ The data is added successfully.



The screenshot shows the Google Cloud Firestore console. The left sidebar has sections for 'Database', 'Indexes', 'Import/Export', 'Disaster Recovery', 'Time-to-live (TTL)', and 'Security Rules'. Under 'Insights', it shows 'Usage' and 'Key Visualizer'. The main area shows the 'tempo' database with a 'creatingfrompython' collection containing a single document 'G7yU50Cn0lF3Xb6kpZ01'. The document details are:

Emp_ID:	EMP001						
Name:	"Sam Curran"						
Address:	"123 Main St"						
contact	<table border="1"><tr><td>Email:</td><td>"sam.curran@super.com"</td></tr><tr><td>Phone:</td><td>7312456890</td></tr><tr><td>position:</td><td>"Cashier"</td></tr></table>	Email:	"sam.curran@super.com"	Phone:	7312456890	position:	"Cashier"
Email:	"sam.curran@super.com"						
Phone:	7312456890						
position:	"Cashier"						

➤ Updating a field in a document.



The screenshot shows the Google Cloud Shell interface. The left sidebar displays a project named 'RAMANAGCP29' with several files listed under the 'firestore' directory: create.py, delete_dummy.py, delete.py, fscommands, gcs_to_firestore.py, import.py, query.py, read.py, sorted.json, super_db.json, and update.py. The 'update.py' file is currently selected in the Explorer view. The main workspace shows a code editor with the following Python script:

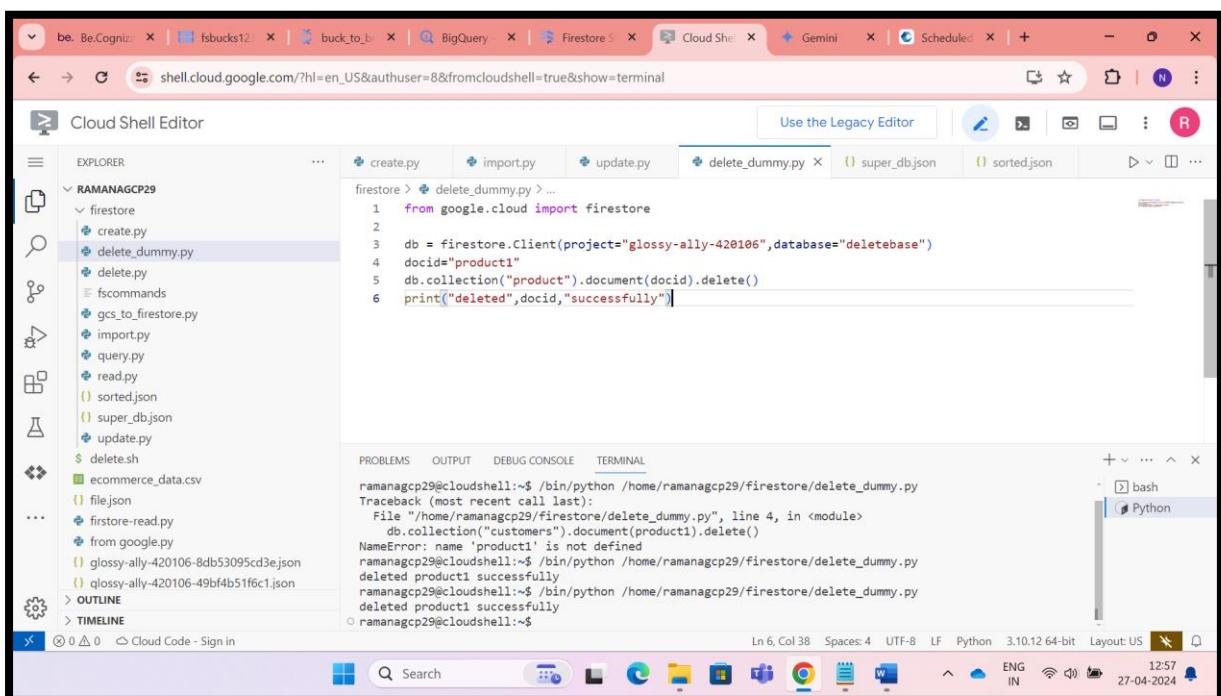
```
firestore > update.py > {} firestore
1   from google.cloud import firestore
2
3   db = firestore.Client(project="glossy-ally-420106",database="mydatabase")
4
5   data={"cities":"sam"}
6
7   doc = db.collection("customers").document("0wgPKzXm5wSnCwfE0XS").update(data)
8   print("Added Successfully")
```

Below the code editor is a terminal window showing the execution of the script:

```
ramanagcp29@cloudshell:~$ /bin/python /home/ramanagcp29/firestore/delete_dummy.py
Traceback (most recent call last):
  File "/home/ramanagcp29/firestore/delete_dummy.py", line 4, in <module>
    db.collection("customers").document(product1).delete()
NameError: name 'product1' is not defined
ramanagcp29@cloudshell:~$ /bin/python /home/ramanagcp29/firestore/delete_dummy.py
deleted product1 successfully
ramanagcp29@cloudshell:~$ /bin/python /home/ramanagcp29/firestore/delete_dummy.py
deleted product1 successfully
ramanagcp29@cloudshell:~$
```

The bottom status bar indicates the session is running on a 3.10.12 64-bit machine with English input and a timestamp of 12:56 on 27-04-2024.

➤ Deleting a document inside a collection.



The screenshot shows the Google Cloud Shell interface. The left sidebar displays a project named 'RAMANAGCP29' with several files listed under the 'firestore' directory: create.py, delete_dummy.py, delete.py, fscommands, gcs_to_firestore.py, import.py, query.py, read.py, sorted.json, super_db.json, and update.py. The 'delete_dummy.py' file is currently selected in the Explorer view. The main workspace shows a code editor with the following Python script:

```
firestore > delete_dummy.py > ...
1   from google.cloud import firestore
2
3   db = firestore.Client(project="glossy-ally-420106",database="deletebase")
4   docid="product1"
5   db.collection("product").document(docid).delete()
6   print("deleted",docid,"successfully")
```

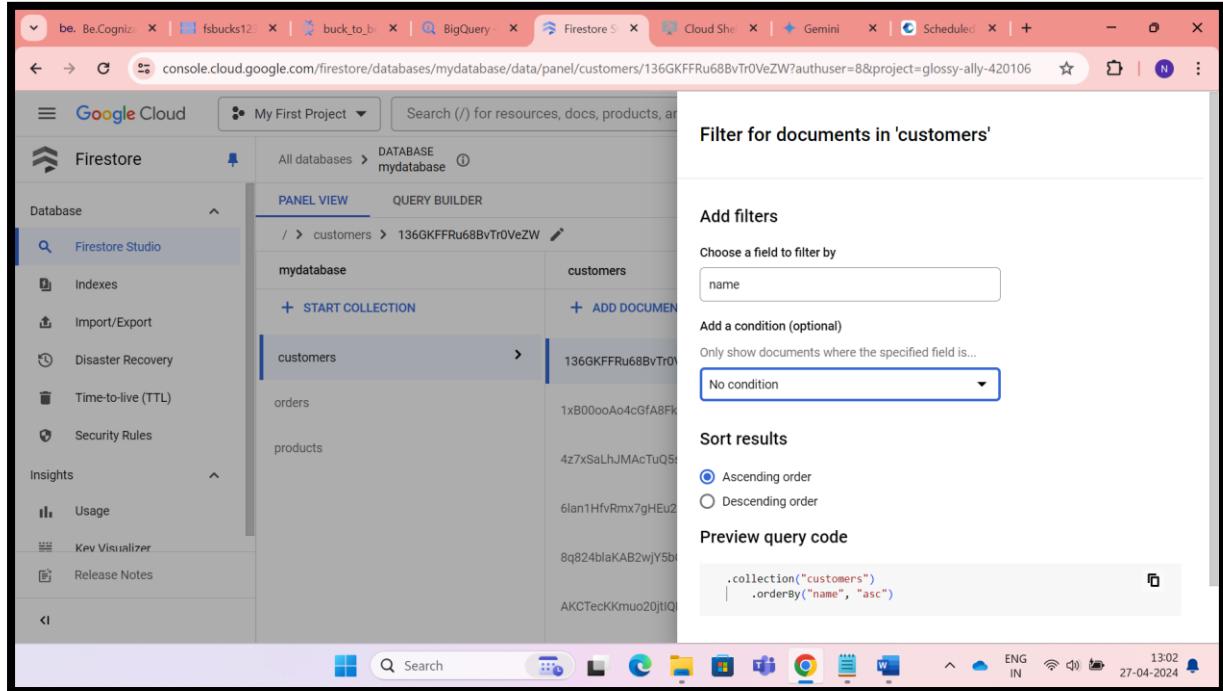
Below the code editor is a terminal window showing the execution of the script:

```
ramanagcp29@cloudshell:~$ /bin/python /home/ramanagcp29/firestore/delete_dummy.py
Traceback (most recent call last):
  File "/home/ramanagcp29/firestore/delete_dummy.py", line 4, in <module>
    db.collection("customers").document(product1).delete()
NameError: name 'product1' is not defined
ramanagcp29@cloudshell:~$ /bin/python /home/ramanagcp29/firestore/delete_dummy.py
deleted product1 successfully
ramanagcp29@cloudshell:~$ /bin/python /home/ramanagcp29/firestore/delete_dummy.py
deleted product1 successfully
ramanagcp29@cloudshell:~$
```

The bottom status bar indicates the session is running on a 3.10.12 64-bit machine with English input and a timestamp of 12:57 on 27-04-2024.

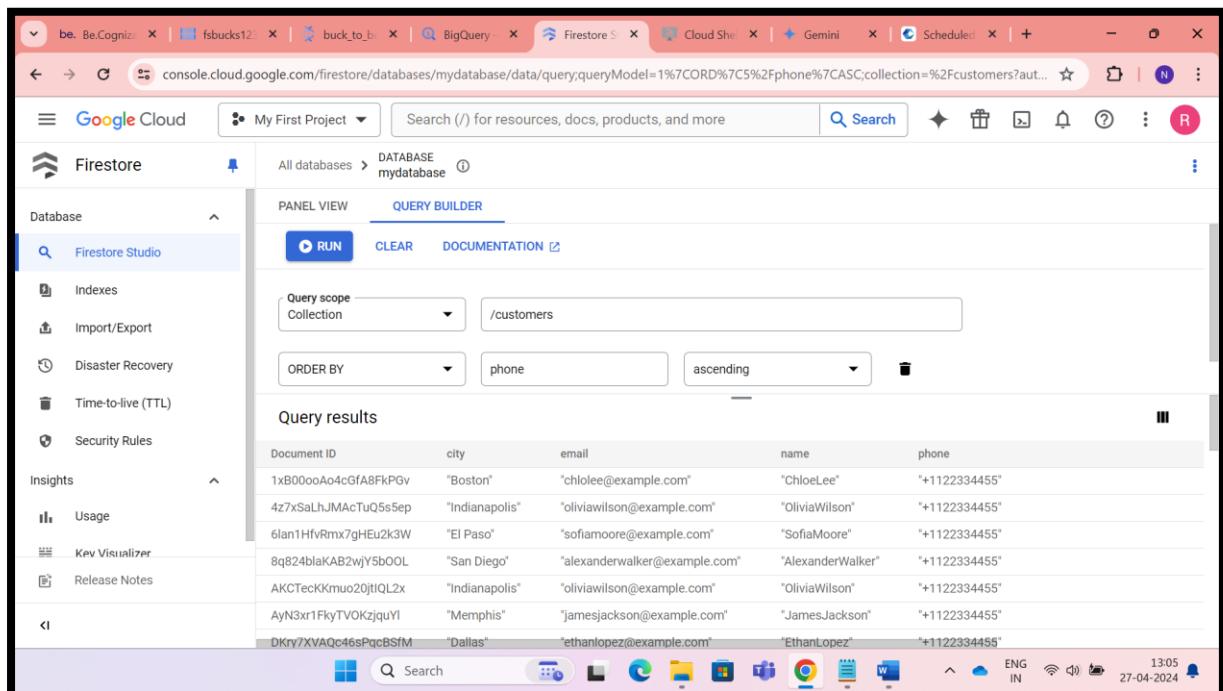
Filtering and querying in Firestore

➤ Filtering



The screenshot shows the Google Cloud Platform interface for Firestore. On the left, the sidebar includes 'Firestore Studio', 'Indexes', 'Import/Export', 'Disaster Recovery', 'Time-to-live (TTL)', 'Security Rules', 'Insights', 'Usage', 'Kv Visualizer', and 'Release Notes'. The main area shows a 'PANEL VIEW' of the database structure under 'mydatabase': 'customers' (selected), 'orders', and 'products'. To the right is the 'QUERY BUILDER' panel titled 'Filter for documents in 'customers''. It contains sections for 'Add filters' (with a 'name' field), 'Add a condition (optional)' (set to 'No condition'), 'Sort results' (set to 'Ascending order'), and 'Preview query code' (showing `.collection("customers") .orderBy("name", "asc")`). The status bar at the bottom indicates '13:02 27-04-2024'.

➤ Querying



The screenshot shows the Google Cloud Platform interface for Firestore. The sidebar is identical to the previous screenshot. The main area shows the 'QUERY BUILDER' panel with 'RUN' selected. It displays a 'Query scope' dropdown set to 'Collection' with '/customers' entered, and an 'ORDER BY' dropdown set to 'phone' with 'ascending' selected. Below this is the 'Query results' section, which lists document data:

Document ID	city	email	name	phone
1xB00ooAo4cGfA8FkPGV	"Boston"	"chlolee@example.com"	"ChloeLee"	"+1122334455"
4z7xSaLhJMACtUQ55ep	"Indianapolis"	"oliviawilson@example.com"	"OliviaWilson"	"+1122334455"
6lan1HfvRmx7gHeu2k3W	"El Paso"	"sofiamoore@example.com"	"SofiaMoore"	+1122334455"
8q824blaKAB2wjY5b0OL	"San Diego"	"alexanderwalker@example.com"	"AlexanderWalker"	+1122334455"
AKCTecKKmuo20jtQL2x	"Indianapolis"	"oliviawilson@example.com"	"OliviaWilson"	+1122334455"
AyN3xr1FkyTVOKzjquYI	"Memphis"	"jamesjackson@example.com"	"JamesJackson"	+1122334455"
DKry/XVAQc46sPqcBSfM	"Dallas"	"ethanlopez@example.com"	"EthanLopez"	+1122334455"

The status bar at the bottom indicates '13:05 27-04-2024'.

➤ Indexing

The screenshot shows the Google Cloud Firestore console interface. The left sidebar has sections for Database (Firestore Studio, Indexes, Import/Export, Disaster Recovery, Time-to-live (TTL), Security Rules), Insights (Usage, Key Visualizer, Release Notes), and a bottom section for Help and Support.

The main content area is titled "Indexes" and shows two tabs: "COMPOSITE" (selected) and "SINGLE FIELD". Below the tabs, a sub-section titled "Composite indexes" includes a "+ CREATE INDEX" button and a "Filter" section with dropdowns for "Collection ID", "Fields", and "Query scope". A message at the bottom says "Build the indexes you need to power your compound queries" and "Run your desired query in your app code to get a link to generate the required index, or create one here."

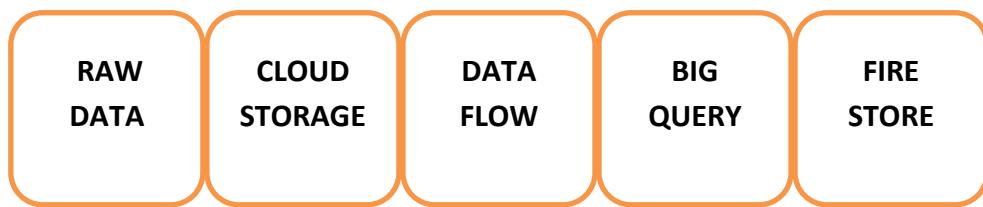
The browser's address bar shows the URL: `console.cloud.google.com/firestore/databases/mydatabase/indexes/composite?authuser=8&project=glossy-ally-420106`.

IMPLEMENTATION :2

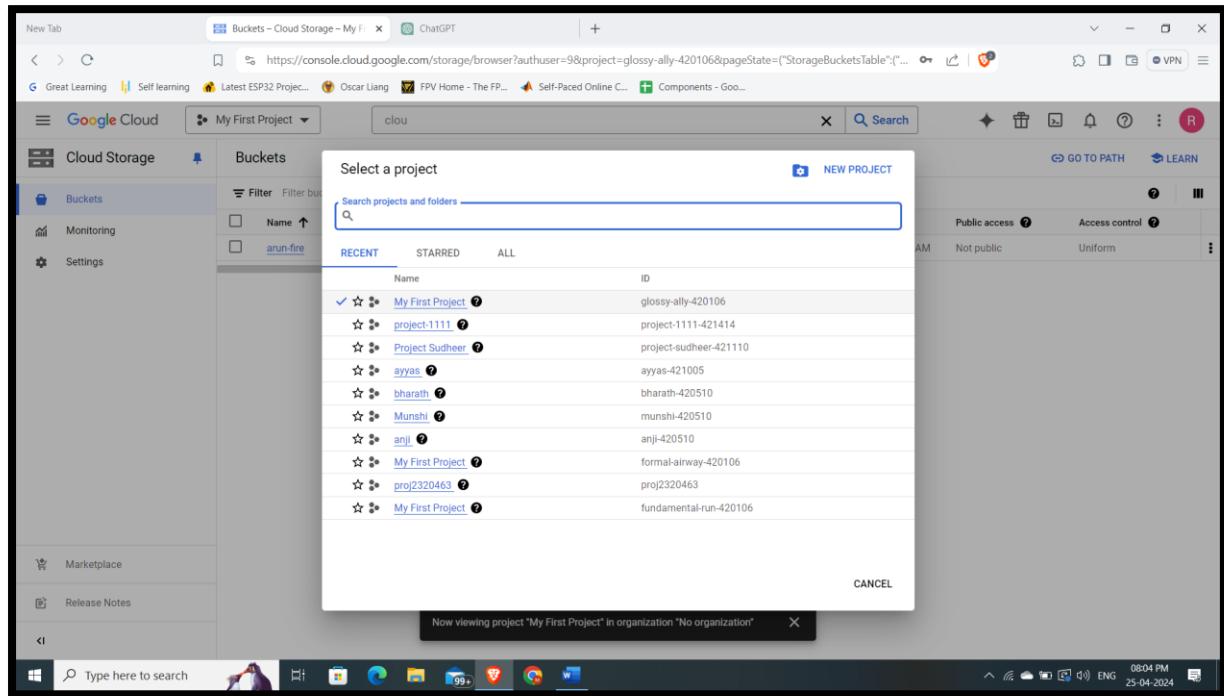
➤ Firestore Integration with Other GCP Services

Here in the implementation part 2, I have integrated some of the GCP services and used them to do some kind of data analysis.

I have used the ecommerce data for our data analysis.



➤ Create a Project.



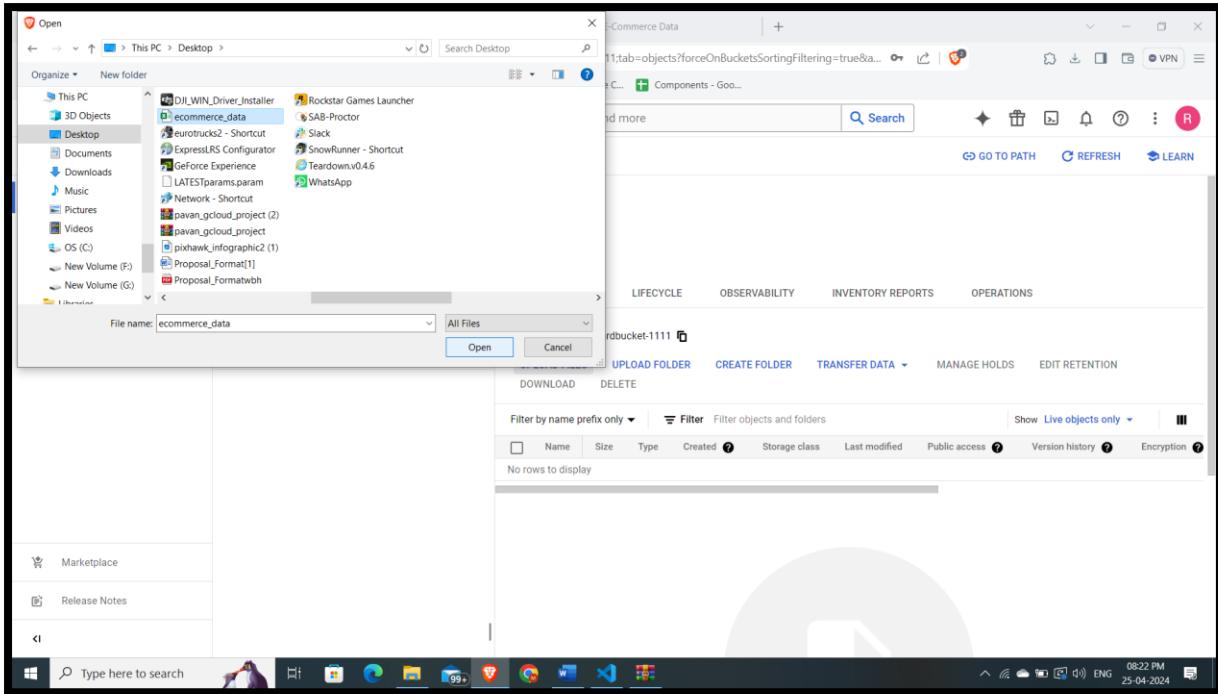
➤ Create Google cloud storage bucket.

The screenshot shows the 'Create a bucket' wizard in the Google Cloud Storage interface. The left sidebar shows 'Buckets', 'Monitoring', and 'Settings'. The main area has five steps: 1. Name your bucket (standardbucket-1111), 2. Choose where to store your data (Location: asia-south1 (Mumbai), Location type: Region), 3. Choose a storage class for your data (Default storage class: Standard), 4. Choose how to control access to objects (Public access prevention: On, Access control: Uniform), and 5. Choose how to protect object data (Soft delete policy: Enabled). A 'Good to know' section includes 'Location pricing' information. A table shows the current configuration: Region / Standard, with a cost of \$0.023 per GB-month for asia-south1 (Mumbai). A 'Data protection' section includes an 'Object retention period' field set to 7 days. The bottom right shows system status: 08:05 PM, ENG, 25-04-2024.

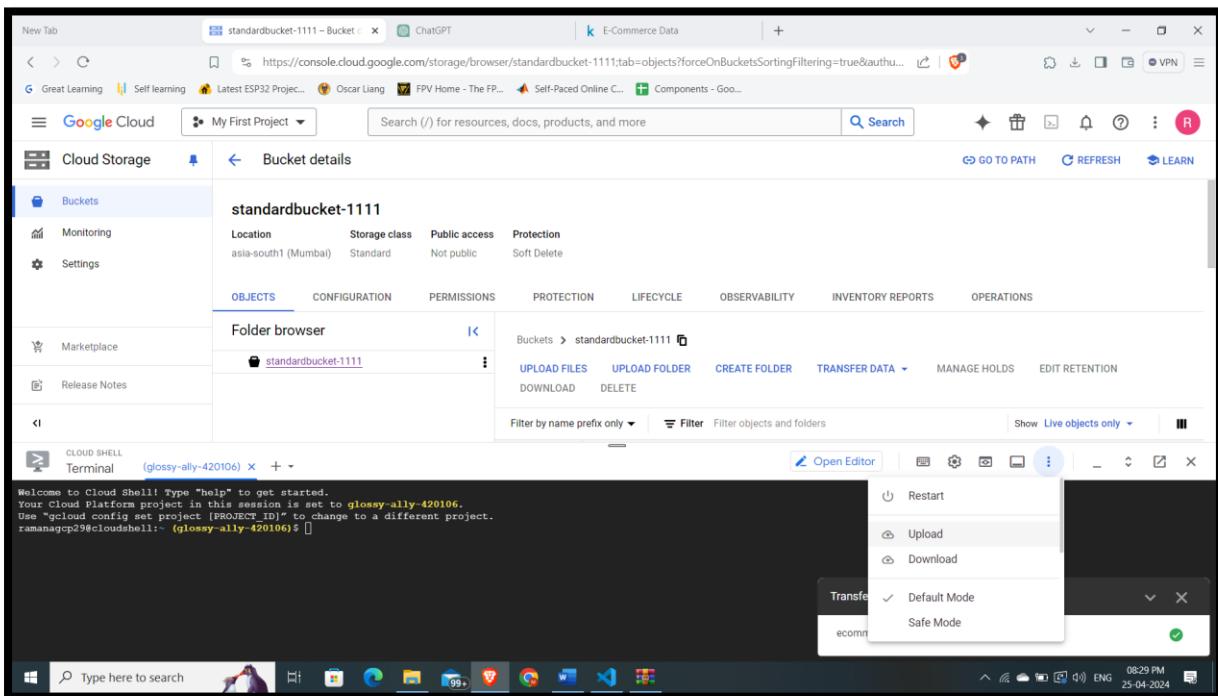
➤ I have created a standard bucket where we can upload our raw data.

The screenshot shows the 'Bucket details' page for 'standardbucket-1111'. The left sidebar shows 'Buckets', 'Monitoring', and 'Settings'. The main area displays bucket metadata: Location (asia-south1 (Mumbai)), Storage class (Standard), Public access (Not public), and Protection (Soft Delete). Below this are tabs for OBJECTS, CONFIGURATION, PERMISSIONS, PROTECTION, LIFECYCLE, OBSERVABILITY, INVENTORY REPORTS, and OPERATIONS. The OBJECTS tab shows a 'Folder browser' with a single folder 'standardbucket-1111'. Action buttons include UPLOAD FILES, UPLOAD FOLDER, CREATE FOLDER, TRANSFER DATA, DOWNLOAD, and DELETE. A filter bar allows filtering by name, type, and storage class. A message at the bottom says 'Created bucket standardbucket-1111'. The bottom right shows system status: 08:07 PM, ENG, 25-04-2024.

- Upload the raw data file from local machine to bucket.



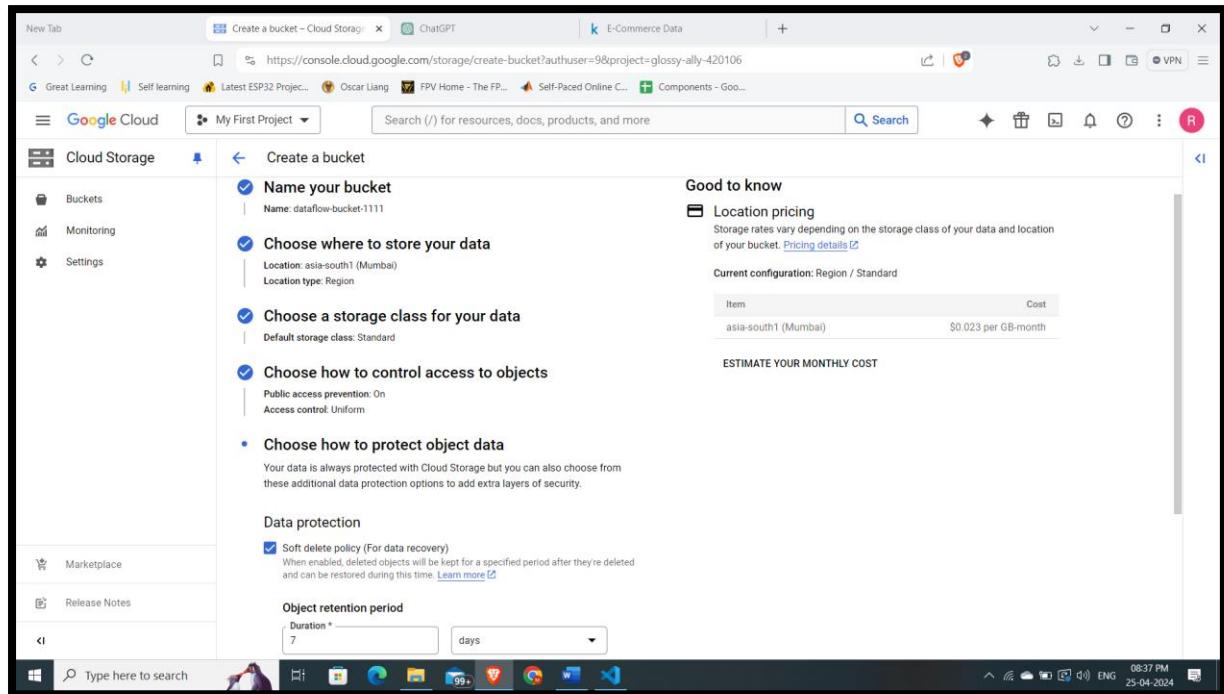
- We can also upload file using GSUTIL commands(optional).



Command to copy file from CLI to GCS bucket.

```
gsutil cp /home/ramanagcp29/ecommerce_data.csv gs://standardbucket-1111/
```

- Let's create another bucket where we will store files related to DATAFLOW.

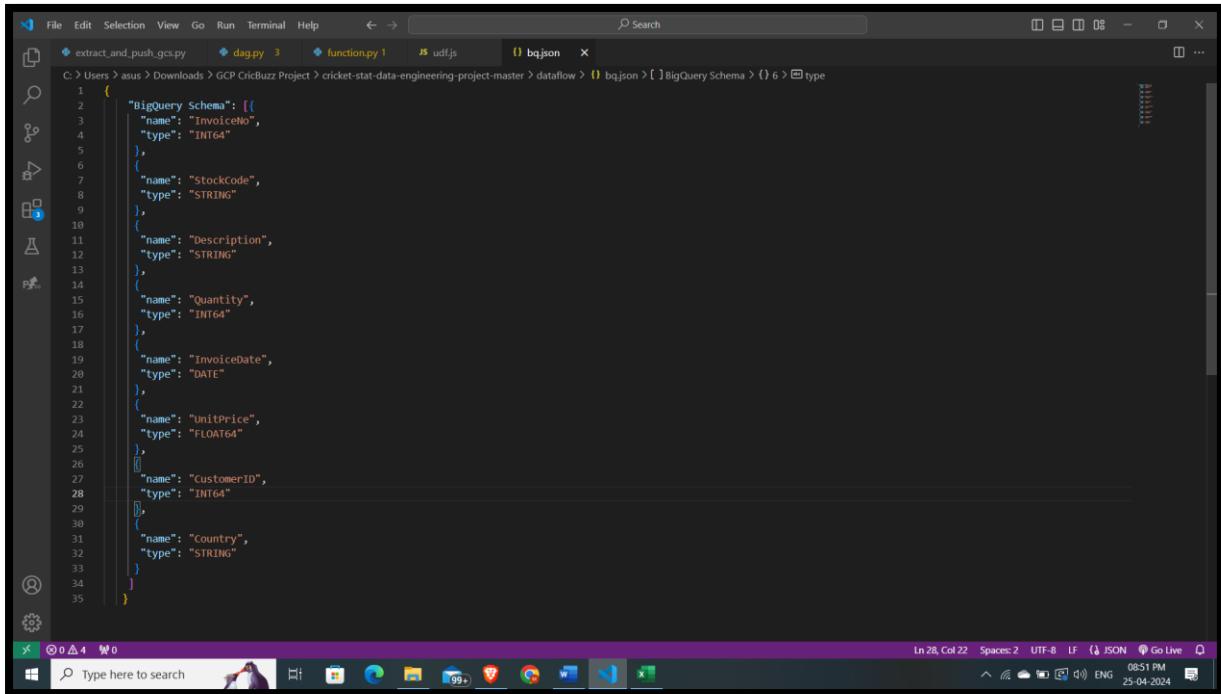


- Creating a UDF function which defines the fields in our raw data which we will be using for analytical purpose.

The screenshot shows a code editor with multiple tabs open. The active tab is 'udf.js' containing the following JavaScript code:

```
C: > Users > asus > Downloads > GCP CricBuzz Project > cricket-stat-data-engineering-project-master > dataflow > udf.js > transform
1  function transform(line) {
2    var values = line.split(',');
3    var obj = new Object();
4    obj.InvoiceNo = values[0];
5    obj.StockCode = values[1];
6    obj.Description = values[2];
7    obj.Quantity = values[3];
8    obj.InvoiceDate = values[4];
9    obj.UnitPrice = values[5];
10   obj.CustomerID = values[6];
11   obj.Country = values[7];
12   var jsonString = JSON.stringify(obj);
13   return jsonString;
14 }
```

- Defining the datatypes of the raw data, which will be the schema for the BIGQUERY in JSON file format.

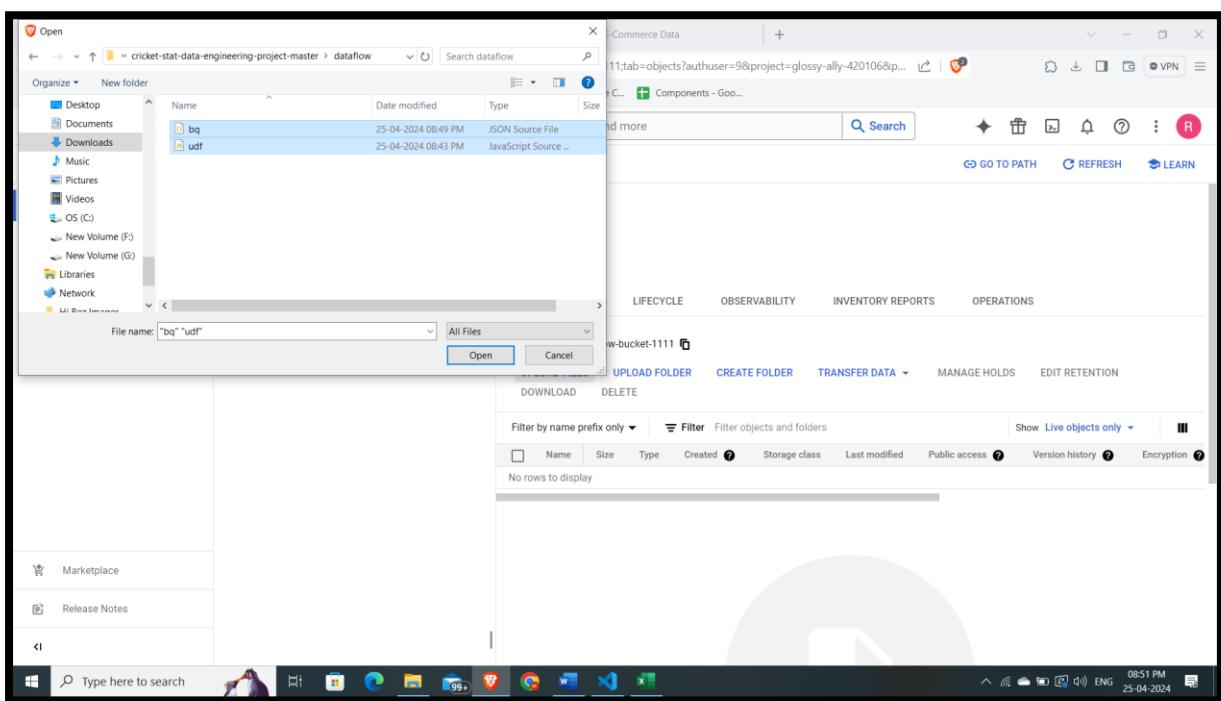


```

1  {
2      "BigQuery Schema": [
3          {
4              "name": "InvoiceNo",
5              "type": "INT64"
6          },
7          {
8              "name": "StockCode",
9              "type": "STRING"
10         },
11         {
12             "name": "Description",
13             "type": "STRING"
14         },
15         {
16             "name": "Quantity",
17             "type": "INT64"
18         },
19         {
20             "name": "InvoiceDate",
21             "type": "DATE"
22         },
23         {
24             "name": "UnitPrice",
25             "type": "FLOAT64"
26         }
27     ],
28     "CustomerID": "INT64"
29 ],
30     {
31         "name": "Country",
32         "type": "STRING"
33     }
34 ]
35

```

- Uploading data to the dataflow bucket.



- Uploading required files successfully.

The screenshot shows the Google Cloud Storage 'Bucket details' page for 'dataflow-bucket-1111'. The bucket is located in 'asia-south1 (Mumbai)' with a 'Standard' storage class, 'Not public' public access, and 'Soft Delete' protection. The 'OBJECTS' tab is selected, displaying a 'Folder browser' for 'dataflow-bucket-1111'. Two files are listed: 'bq.json' (548 B, application/json) and 'udf.js' (408 B, text/javascript). A success message at the bottom indicates '2 files successfully uploaded'.

- Create a database in BIGQUERY and an empty table.

The screenshot shows the Google BigQuery 'Explorer' interface. The left sidebar shows a tree view of resources under 'glossy-ally-420106'. Under the 'ecommerce_dataset' node, there is a single table named 'ecommerce_table'. The main area is titled 'Untitled query' with a count of '1'.

➤ Create job from template in DATAFLOW.

The screenshot shows the 'Create job from template' page in the Google Cloud Dataflow interface. On the left, there are several configuration sections:

- Job name ***: data-flow-gcs-bigquery
- Regional endpoint ***: asia-south1 (Mumbai)
- Dataflow template ***: Text Files on Cloud Storage to BigQuery
- Required Parameters** section:
 - Cloud Storage Input File(s) *: gs://standardbucket-1111/ecommerce_data.csv
 - Cloud Storage location of your BigQuery schema file, described as a JSON *: gs://dataflow-bucket-1111/bq.json

On the right, there is a large 'Additional information' panel containing a flowchart of the pipeline. The flowchart starts with 'Read from source' (Text Files on Cloud Storage), followed by 'JavaScriptT...Javascript' (a UDF), then 'BigQueryCon...ToTableRow', and finally 'Insert into Bigquery'.

➤ Give all the required input output file and folder paths.

The screenshot shows the 'Create job from template' page with the following completed fields:

- Required Parameters** section:
 - Cloud Storage Input File(s) *: gs://standardbucket-1111/ecommerce_data.csv
 - Cloud Storage location of your BigQuery schema file, described as a JSON *: gs://dataflow-bucket-1111/bq.json
 - BigQuery output table *: glossy-ally-420106:ecommerce_dataset.ecommerce_table
 - Temporary directory for BigQuery loading process *: gs://dataflow-bucket-1111
 - Temporary location *: gs://dataflow-bucket-1111/temp
- Encryption** section:
 - Google-managed encryption key
No configuration required
 - Customer-managed encryption key (CMK)
Manage via [Google Cloud Key Management Service](#)

- Run the job and check for successful completion.

The screenshot shows the Google Cloud Dataflow interface. On the left, the sidebar includes 'Dataflow' (selected), 'Overview', 'Monitoring', 'Jobs' (selected), 'Pipelines', 'Workbench', 'Snapshots', and 'SQL Workspace'. Below the sidebar is a 'Release Notes' section. The main area has tabs for 'JOB GRAPH' (selected), 'EXECUTION DETAILS', and 'JOB METRICS'. The 'JOB GRAPH' tab displays a vertical flow of four stages: 'Read from source' (Succeeded, 5 sec, 2 of 2 stages succeeded), 'JavascriptT...Javascript' (Succeeded, 13 sec, 1 of 1 stage succeeded), 'BigQueryCon...ToTableRow' (Succeeded, 6 sec, 1 of 1 stage succeeded), and 'Insert into Bigquery' (Succeeded). To the right of the graph is the 'Job info' panel, which provides detailed information about the job, such as 'Job name: data-flow-gcs-bigquery1', 'Job ID: 2024-04-25_10_05_20-3585772906144489911', 'Job type: Batch', 'Job status: Succeeded', and 'SDK version: Apache Beam SDK for Java 2.55.1'. Below the job info is the 'Resource metrics' panel, which shows CPU usage and memory consumption.

- The data is successfully loaded into the bigquery table with the desired schema.

The screenshot shows the Google Cloud BigQuery interface. On the left, the 'Explorer' sidebar lists 'glossy-ally-420106' (selected), 'Queries', 'Notebooks', 'Data canvases', 'External connections', and 'ecommerce_dataset' (selected), which contains 'ecommerce_table'. Below the sidebar is a 'SUMMARY' section for 'ecommerce_table', showing it was last modified on April 25, 2024, at 10:40:54 PM UTC+5:30, and is located in 'asia-south1'. The main area is titled 'ecommerce_table' and shows the 'SCHEMA' tab selected. It displays a table of columns with their properties:

Field name	Type	Mode	Key	Collation	Default Value	Policy Tags	Description
InvoiceNo	STRING	NULLABLE	-	-	-	-	-
StockCode	STRING	NULLABLE	-	-	-	-	-
Description	STRING	NULLABLE	-	-	-	-	-
Quantity	STRING	NULLABLE	-	-	-	-	-
UnitPrice	STRING	NULLABLE	-	-	-	-	-
CustomerID	STRING	NULLABLE	-	-	-	-	-
Country	STRING	NULLABLE	-	-	-	-	-

At the bottom of the schema editor are buttons for 'EDIT SCHEMA' and 'VIEW ROW ACCESS POLICIES'.

- This is the preview of the data in table with all the columns and rows.

The screenshot shows the Google Cloud BigQuery interface. In the left sidebar, under 'My First Project', there's a tree view of resources. Under 'ecommerce_dataset', the 'ecommerce_table' is selected. The main area displays the schema and a preview of the data. The schema includes columns: Row, InvoiceNo, StockCode, Description, Quantity, UnitPrice, and CustomerID. The preview shows 17 rows of data with various values for these columns. At the bottom, there are navigation controls for results per page (200) and a refresh button.

- Query to cast the datatypes, string to integer and created new table.

The screenshot shows the Google Cloud BigQuery interface. In the left sidebar, 'BigQuery Studio' is selected. An 'Untitled query' tab is open, containing the following SQL code:

```

1 -- Create a new table with the updated schema
2 CREATE OR REPLACE TABLE `glossy-alley-420106.ecommerce_dataset.ecommerce_table1` AS
3 SELECT
4   InvoiceNo,
5   StockCode,
6   Description,
7   Quantity,
8   SAFE_CAST(UnitPrice AS INT64) AS UnitPrice,
9   CustomerID,
10  Country
11 FROM
12  `glossy-alley-420106.ecommerce_dataset.ecommerce_table`;
13

```

The 'Query results' section shows a message: 'This statement created a new table named ecommerce_table1.' There is a 'GO TO TABLE' button. The rest of the interface shows the same resource tree and navigation elements as the previous screenshot.

- Updated schema of UnitPrice field with integer datatype.

The screenshot shows the Google Cloud BigQuery schema editor. On the left, the navigation pane includes sections like Analysis, Migration, and Administration. The main area displays the schema for the dataset 'ecommerce_dataset' and table 'ecommerce_table1'. The schema table has columns for Field name, Type, Mode, Key, Collation, Default Value, Policy Tags, and Description. The 'UnitPrice' field is now defined as an INTEGER type. Other fields like InvoiceNo, StockCode, Description, Quantity, CustomerID, and Country remain as STRING types.

Field name	Type	Mode	Key	Collation	Default Value	Policy Tags	Description
InvoiceNo	STRING	NULLABLE	-	-	-	-	-
StockCode	STRING	NULLABLE	-	-	-	-	-
Description	STRING	NULLABLE	-	-	-	-	-
Quantity	STRING	NULLABLE	-	-	-	-	-
UnitPrice	INTEGER	NULLABLE	-	-	-	-	-
CustomerID	STRING	NULLABLE	-	-	-	-	-
Country	STRING	NULLABLE	-	-	-	-	-

- Data cleansing by removing null values, and count, sum operation on the above table.

The screenshot shows the Google Cloud BigQuery query editor. A query titled 'Untitled query' is displayed in the editor. The SQL code removes rows where any column is null and then performs a group by operation on the 'Country' column to calculate the total transaction count and revenue. The results table shows one row for Germany.

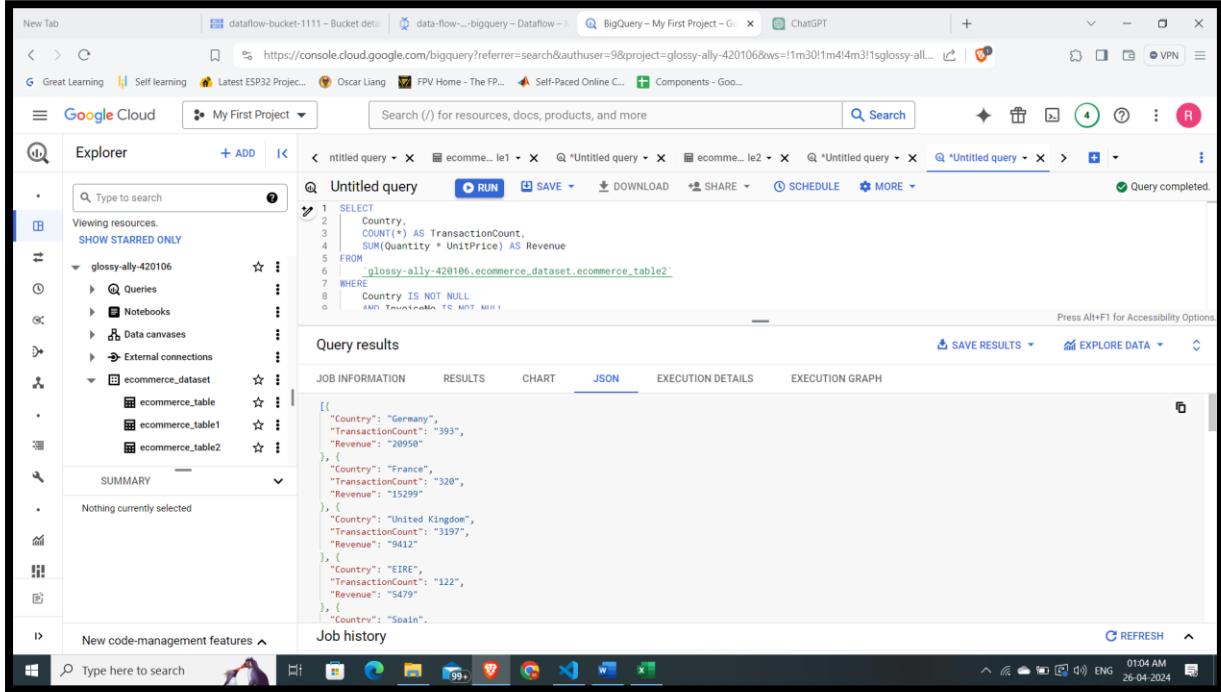
```

1 SELECT
2   Country,
3   COUNT(*) AS TransactionCount,
4   SUM(Quantity * UnitPrice) AS Revenue
5 FROM
6   `glossy-ally-420106.ecommerce_dataset.ecommerce_table2`
7 WHERE
8   Country IS NOT NULL
9   AND InvoiceNo IS NOT NULL
10  AND StockCode IS NOT NULL
11  AND Description IS NOT NULL
12  AND Quantity IS NOT NULL
13  AND UnitPrice IS NOT NULL
14  AND CustomerID IS NOT NULL
15 GROUP BY
16   Country
17 ORDER BY
18   Revenue DESC;
19

```

Row	Country	TransactionCount	Revenue
1	Germany	393	20950

- The tables in bigquery also gives the data in JSON format.



The screenshot shows the Google Cloud BigQuery interface. On the left, the Explorer sidebar displays a project named "glossy-ally-420106" with a dataset "ecommerce_dataset" containing three tables: "ecommerce_table", "ecommerce_table1", and "ecommerce_table2". The main area shows an "Untitled query" with the following SQL code:

```

1 SELECT
2     Country,
3     COUNT(*) AS TransactionCount,
4     SUM(Quantity * UnitPrice) AS Revenue
5 FROM `glossy-ally-420106.ecommerce_dataset.ecommerce_table2`
6 WHERE
7     Country IS NOT NULL
8     AND TransactionCount > 0

```

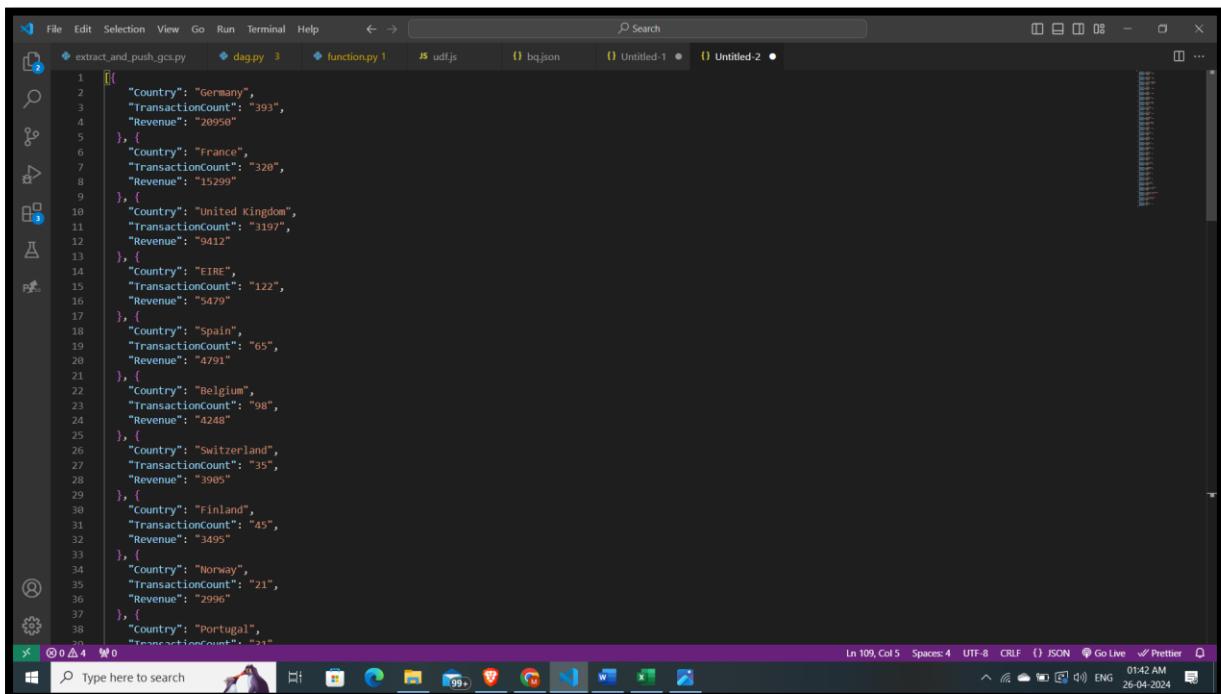
The "Query results" section displays the output in JSON format, listing countries with their transaction counts and revenue:

```

[{"Country": "Germany", "TransactionCount": "393", "Revenue": "20950"}, {"Country": "France", "TransactionCount": "320", "Revenue": "15299"}, {"Country": "United Kingdom", "TransactionCount": "3197", "Revenue": "9412"}, {"Country": "EIRE", "TransactionCount": "122", "Revenue": "5479"}, {"Country": "Spain", "TransactionCount": "65", "Revenue": "4791"}, {"Country": "Belgium", "TransactionCount": "98", "Revenue": "4248"}, {"Country": "Switzerland", "TransactionCount": "35", "Revenue": "3905"}, {"Country": "Finland", "TransactionCount": "45", "Revenue": "3495"}, {"Country": "Norway", "TransactionCount": "21", "Revenue": "2996"}, {"Country": "Portugal", "TransactionCount": "21", "Revenue": "2996"}]

```

- JSON data copied to local.



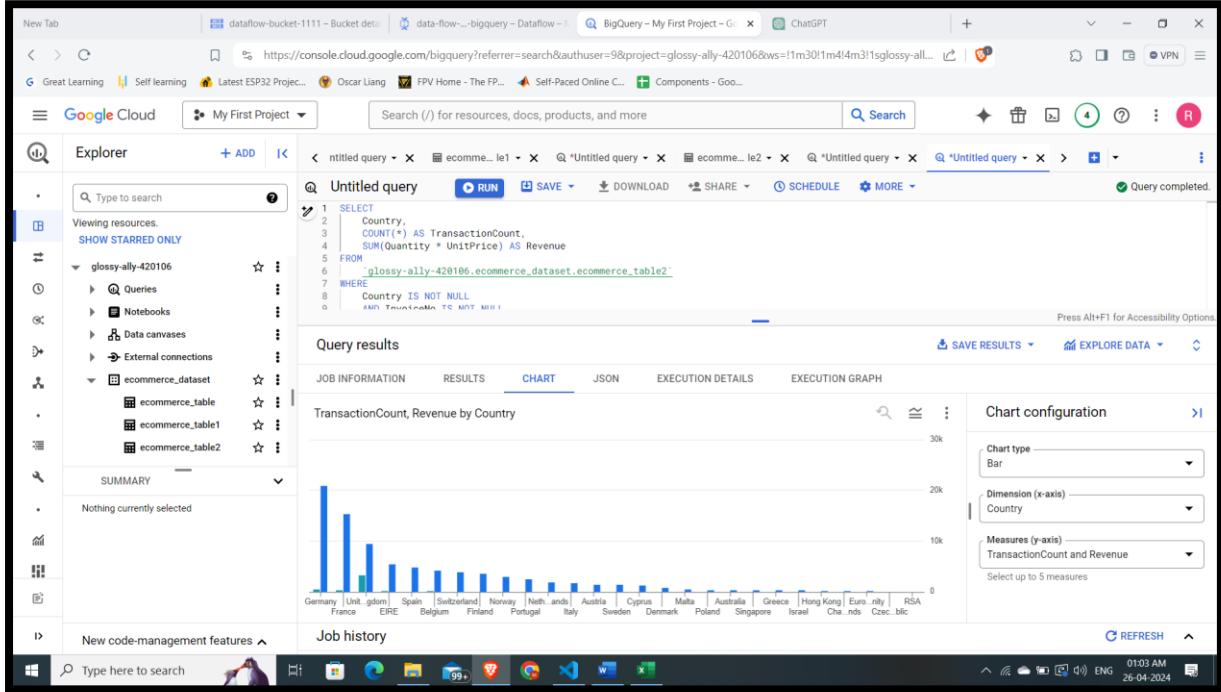
The screenshot shows a code editor window with several tabs open. The active tab contains the JSON data copied from the previous screenshot. The code is identical to the one shown in the BigQuery results:

```

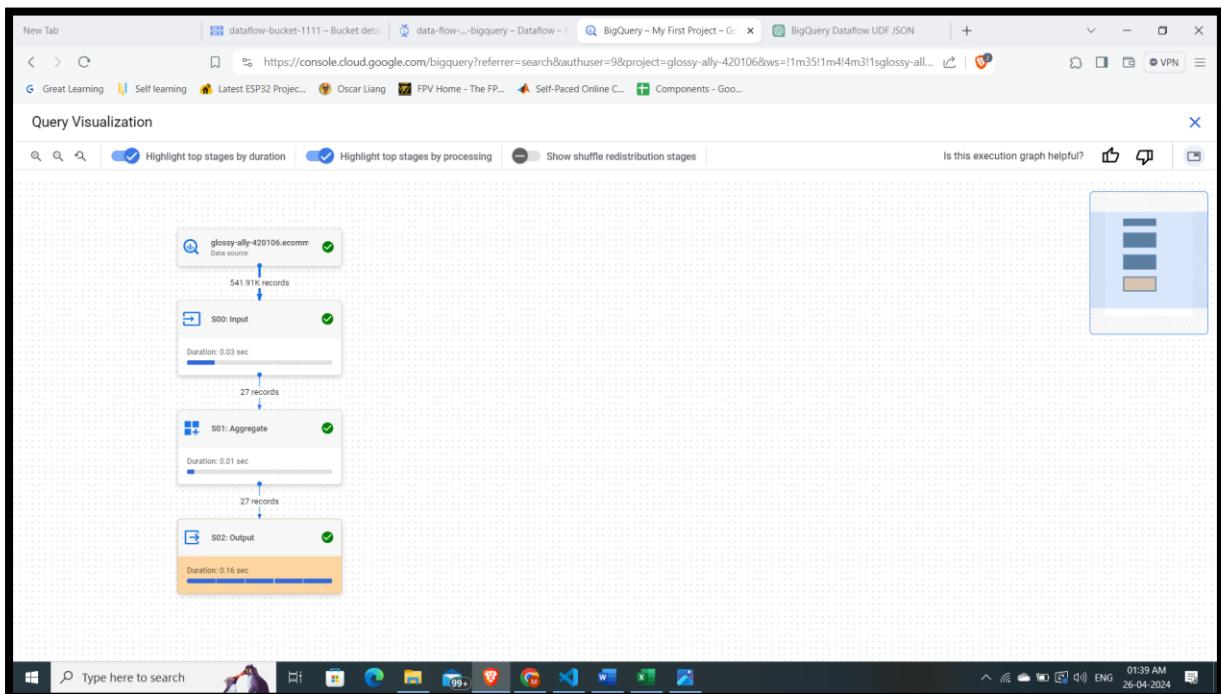
[{"Country": "Germany", "TransactionCount": "393", "Revenue": "20950"}, {"Country": "France", "TransactionCount": "320", "Revenue": "15299"}, {"Country": "United Kingdom", "TransactionCount": "3197", "Revenue": "9412"}, {"Country": "EIRE", "TransactionCount": "122", "Revenue": "5479"}, {"Country": "Spain", "TransactionCount": "65", "Revenue": "4791"}, {"Country": "Belgium", "TransactionCount": "98", "Revenue": "4248"}, {"Country": "Switzerland", "TransactionCount": "35", "Revenue": "3905"}, {"Country": "Finland", "TransactionCount": "45", "Revenue": "3495"}, {"Country": "Norway", "TransactionCount": "21", "Revenue": "2996"}, {"Country": "Portugal", "TransactionCount": "21", "Revenue": "2996"}]

```

- The Data also can be viewed in chart or graphical format.



- Query Visualization.



➤ Exporting table back to Google cloud storage.

The screenshot shows the Google Cloud BigQuery interface. On the left, the navigation pane includes sections like Analysis, Migration, Administration, and Partner Center. The main area displays the 'ecommerce_sorted_table' from the 'ecommerce_dataset'. A context menu is open over the table, with the 'EXPORT' option highlighted. Other options in the menu include 'Explore with Sheets', 'Export to GCS', and 'Scan with Sensitive Data Protection'. The table preview shows columns for Row, Country, TransactionCount, and Revenue, with data for 17 rows of various European countries and their transaction counts and revenues.

➤ Choosing export format and location.

The screenshot shows the Google Cloud BigQuery interface with the 'ecommerce_sorted_table' selected. A modal dialog titled 'Export table to Google Cloud Storage' is open. In the dialog, the 'GCS Location' field is set to 'bgq_to_gcs/jason_folder/ecommerce_json'. The 'Export format' is set to 'JSON (Newline delimited)'. The 'Compression' dropdown is set to 'None'. At the bottom left of the dialog, there is a 'SAVE' button. A message at the bottom of the screen says 'Created bucket bgq_to_gcs'. The background shows the same BigQuery interface as the previous screenshot.

➤ New table data from Bigquery is saved here.

The screenshot shows the Google Cloud Storage 'Bucket details' page for the 'bgq_to_gcs' bucket. The bucket is located in 'asia-south1 (Mumbai)' with a 'Standard' storage class, 'Not public' public access, and 'Soft Delete' protection. The 'OBJECTS' tab is selected, displaying a 'Folder browser' for the 'bgq_to_gcs' folder. Inside, there is a single file named 'ecommerce_json' with a size of 1.7 KB, type 'application/octet-stream', and a creation date of April 26, 2024, at 2:34:55 AM. The storage class is listed as 'Standard'. A message box at the bottom left says 'Requested entity was not found.'