Healthcare Data Analysis

Objective:
I builded a Spark application that will process a daily CSV file from a HDFS folder and perform certain transformations on it, and then store the transformed data in a Cassandra table.

Tools Used:
1. Python3 – Microsoft VScode
2. Databricks
3. DataStax Astra (Cassandra DB)

Files Attached:
1. stage_healthcare_analysis– Pyspark File that creates stage tables for daily load.
2. target_healthcare_analysis– Pyspark File that creates final target tables for daily load.

Process and File Descriptions:

Step 1:

So first created csv dataset using a python mock_data_generator file and uploaded that to gcs bucket input folder. Then I created a spark job that takes the daily file from the healthcare_analysis bucket and input folder. I made sure that there is authentication between databricks and GCP cloud storage, by placing the keys in the dbfs location.

```
# Path to the service account JSON key in DBFS
service_account_path = "/dbfs/FileStore/shared_uploads/auth/noob2_bootcamp_407704_058a42626b1b.json"


# Configure Spark to use the service account JSON key for GCS authentication
spark.conf.set("fs.gs.auth.service.account.json.keyfile", service_account_path)


# GCS bucket details
bucket_name = "healthcare_analysis"
data_directory = f"gs://{bucket_name}/input/"
archive_directory = f"gs://{bucket_name}/archive/"
```

```
    # Read all CSV files from the specified GCS directory
    df = spark.read.csv(data_directory, inferSchema=True, header=True)

    df.show()
```

```
+----------+---+------+--------------+---------------------+--------------+
|patient_id|age|gender|diagnosis_code|diagnosis_description|diagnosis_date|
+----------+---+------+--------------+---------------------+--------------+
|        P1| 45|     M|          H234|  High Blood Pressure|    2023-08-01|
|        P2| 32|     F|          D123|             Diabetes|    2023-08-01|
|        P3| 39|     F|          H234|  High Blood Pressure|    2023-08-01|
|        P4| 40|     F|          C345|               Cancer|    2023-08-01|
|        P5| 52|     M|          H234|  High Blood Pressure|    2023-08-01|
|        P6| 43|     F|          C345|               Cancer|    2023-08-01|
|        P7| 51|     M|          D123|             Diabetes|    2023-08-01|
|        P8| 67|     F|          H234|  High Blood Pressure|    2023-08-01|
|        P9| 32|     F|          D123|             Diabetes|    2023-08-01|
|       P10| 63|     M|          H234|  High Blood Pressure|    2023-08-01|
|       P11| 61|     M|          C345|               Cancer|    2023-08-01|
|       P12| 67|     F|          D123|             Diabetes|    2023-08-01|
|       P13| 42|     F|          H234|  High Blood Pressure|    2023-08-01|
|       P14| 65|     F|          H234|  High Blood Pressure|    2023-08-01|
|       P15| 61|     F|          D123|             Diabetes|    2023-08-01|
|       P16| 38|     F|          D123|             Diabetes|    2023-08-01|
|       P17| 69|     F|          H234|  High Blood Pressure|    2023-08-01|
|       P18| 62|     M|          H234|  High Blood Pressure|    2023-08-01|
|       P19| 38|     M|          D123|             Diabetes|    2023-08-01|
|       P20| 55|     F|          D123|             Diabetes|    2023-08-01|
+----------+---+------+--------------+---------------------+--------------+
only showing top 20 rows
```

Step 2:

I made sure to include data quality checks so that the data is in the correct format.

```
    # Check for null values in each column
    null_counts = df.agg(
        *[sum(col(column).isNull().cast("int")).alias(f"{column}_null_count") for column in df.columns]
    )

    # Check for data types
    data_type_checks = [col(column).cast("string").alias(f"{column}_type_check") for column in df.columns]

    # Apply the data type checks
    df_check = df.select(data_type_checks)

    # Show the results of the checks
    print("Null Counts:")
    null_counts.show()

    print("Data Type Checks:")
    df_check.show()
```

```
Null Counts:
+-------------------+-------------+---------------+---------------------+-----------------------------+---------------------+
|patient_id_null_count|age_null_count|gender_null_count|diagnosis_code_null_count|diagnosis_description_null_count|diagnosis_date_null_count|
+-------------------+-------------+---------------+---------------------+-----------------------------+---------------------+
|                  0|            0|              0|                    0|                            0|                    0|
+-------------------+-------------+---------------+---------------------+-----------------------------+---------------------+
```

Step 3:

I then went ahead and performed the necessary transformations/queries on it, post which I followed the documentation for Datastax AstraDB to generate a connection between databricks and the Cassandra DB. This involved downloading a 'secure bundle' as well as the key/token which was then placed in the DBFS location.

```
# Connecting to CassandraDB using Datastax
# This secure connect bundle is autogenerated when you download your SCB,
# if yours is different update the file name below
cloud_config= {
  'secure_connect_bundle': '/dbfs/FileStore/shared_uploads/secure_connect_healthcare_db.zip'
}

# This token JSON file is autogenerated when you download your token,
# if yours is different update the file name below
with open("/dbfs/FileStore/shared_uploads/healthcare_db_token__1_.json") as f:
    secrets = json.load(f)

CLIENT_ID = secrets["clientId"]
CLIENT_SECRET = secrets["secret"]

auth_provider = PlainTextAuthProvider(CLIENT_ID, CLIENT_SECRET)
cluster = Cluster(cloud=cloud_config, auth_provider=auth_provider)
session = cluster.connect()

row = session.execute("select release_version from system.local").one()
if row:
  print("Cassandra Connection Sucessful")
else:
  print("An error occurred.")

keyspace="healthcare"
table='stage_disease_ratio'
```

```
Cassandra Connection Sucessful
```

Step 4:

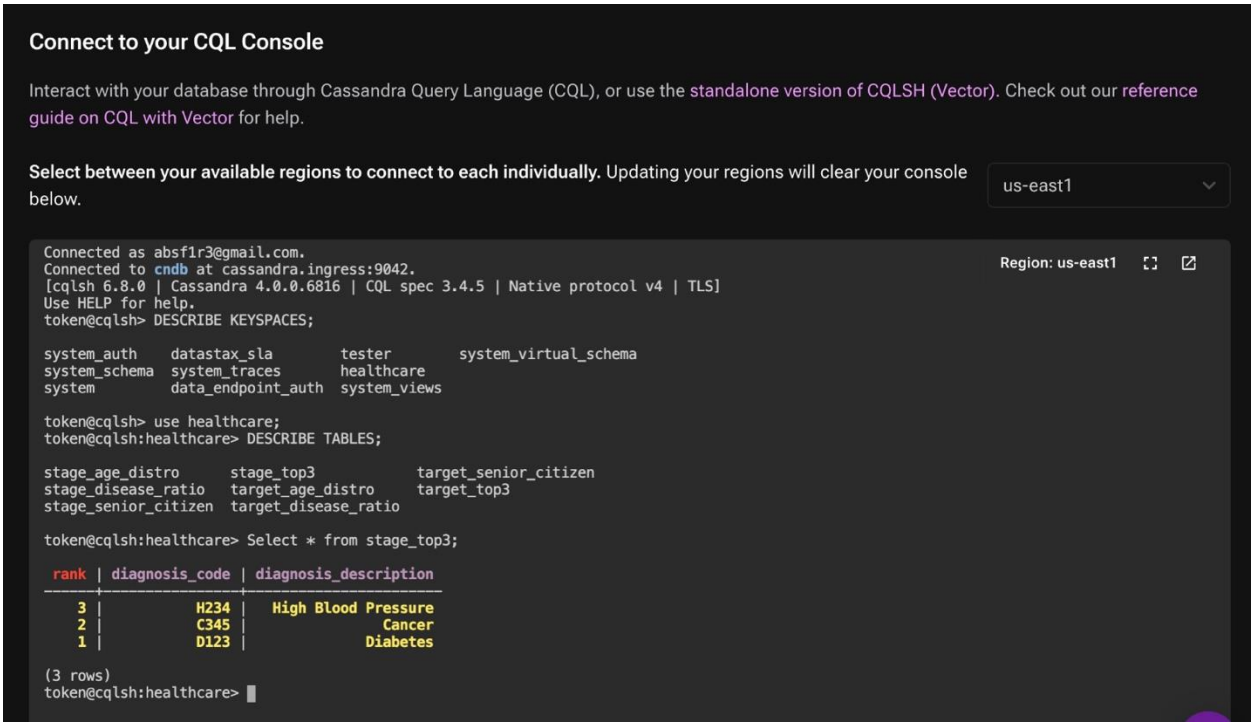
After that using CQL I then checked if there was any table (respective table) in the keyspace. If not I created a new table and then pushed the data into it. If there was an

existing table then I truncated all the data and loaded the new data. (This is like forming a daily staging table)

```
# Check if the table exists
existing_table_query = f"SELECT table_name FROM system_schema.tables WHERE keyspace_name = '{keyspace}' AND table_name = '{table}'"
existing_table_result = session.execute(existing_table_query)

if existing_table_result.one():
    # Table exists, truncate (delete all data)
    truncate_query = f"TRUNCATE TABLE {keyspace}.{table}"
    session.execute(truncate_query)
else:
    # Table does not exist, create it
    create_table_query = f"""
    CREATE TABLE IF NOT EXISTS healthcare.stage_disease_ratio (
        diagnosis_code TEXT PRIMARY KEY,
        diagnosis_description TEXT,
        F_Females INT,
        M_Males INT,
        Gender_Ratio DOUBLE
    )
    """
    session.execute(create_table_query)

# Convert Spark DataFrame to Pandas DataFrame
pandas_df = gender_ratio.toPandas()

# Insert data into Cassandra table
for index, row in pandas_df.iterrows():
    insert_query = f"""
    INSERT INTO healthcare.stage_disease_ratio
    (diagnosis_code, diagnosis_description, F_Females, M_Males, Gender_Ratio)
    VALUES ('{row['diagnosis_code']}', '{row['diagnosis_description']}',
            {row['F_Females']}, {row['M_Males']}, {row['Gender_Ratio']})
    """
    session.execute(insert_query)
```

Step 5:

I also checked using the CQL UI from datastax to see if data is present in the tables. I created stage tables for each of the scenarios and also made sure to archive the input files.



Step 6:

In another script I created target tables for each of the stage tables where data is inserted in the ‘upsert’ mode. The idea is for the target tables, data is moved from the respective stage table→ if no target table exists then a new target table is created and data from the stage table is pushed , else if a target table already exists then upsert is peformed. I made sure to select appropriate keys to match for each of those tables.

```
(3 rows)
token@cqlsh:healthcare> Select * from target_disease_ratio;

 diagnosis_code | diagnosis_description | f_females | gender_ratio | m_males
----------------+-----------------------+-----------+--------------+---------
           C345 |                Cancer |        12 |      1.83333 |      22
           D123 |              Diabetes |        23 |     0.478261 |      11
           H234 |   High Blood Pressure |        18 |     0.777778 |      14

(3 rows)
token@cqlsh:healthcare>
```

Step 7:

I then created a healthcare_processing workflow for the two scripts using databricks. The second job is triggered only when the staging process is finished. This way there is a dependency between the two jobs. I also made sure to implement notifications for any failures.

Challenges:

1. It was not possible to get the Cassandra-spark connector to work (Lots of time spent on trying to get the correct jar/jdbc drivers) and had to resort to the ‘session_execute’ method of loading data row by row.