

## **1. Requirement and Specification**

A Health Care insurance company is facing challenges in enhancing its revenue and understanding the customers. To address these challenges, the company aims to leverage the Big Data Ecosystem to analyze competitors' company data received from various sources, such as web scraping and third-party sources. This analysis will help track customer behavior and conditions, enabling the company to customize offers and calculate royalties for customers who have previously purchased policies, thereby enhancing revenue.

## **2. Project Goals**

The primary goal of this project is to create data pipelines for the Health Care insurance company. These pipelines will enable the company to:

- Analyze customer behaviors.
- Send customized offers to customers.
- Calculate and distribute royalties to past customers.
- Develop appropriate business strategies to enhance revenue.

## **3. Environment**

- **AWS S3:** Data storage for raw and processed data
- **AWS Redshift:** Data warehousing for analysis and reporting
- **Databricks:** Data processing and analysis platform
- **PySpark:** Distributed data processing framework
- **GitHub:** Version control and code repository
- **Jira:** Project management and tracking
- **AWS EMR Studio:** For deployment (optional)

## **4. Requirements**

- **Disease Analysis:** Identify the disease with the maximum number of claims.
- **Subscriber Analysis:** Find subscribers under the age of 30 who subscribe to any subgroup.
- **Group Analysis:** Determine which group has the maximum number of subgroups.
- **Hospital Analysis:** Identify the hospital that serves the most number of patients.
- **Subgroup Analysis:** Find out which subgroup is subscribed to the most.
- **Claims Rejection Analysis:** Calculate the total number of claims that were rejected.
- **Claims Origin Analysis:** Determine the city from which most claims are coming.
- **Policy Type Analysis:** Identify whether subscribers mostly subscribe to government or private policies.

- **Premium Analysis:** Calculate the average monthly premium paid by subscribers to the insurance company.
- **Profitability Analysis:** Find out which group is most profitable.
- **Pediatric Cancer Patients:** List all patients below age 18 who are admitted for cancer.
- **High-Value Cashless Insurance Patients:** List patients who have cashless insurance and total charges greater than or equal to Rs. 50,000.
- **Female Knee Surgery Patients:** List female patients over the age of 40 that have undergone knee surgery in the past year.

## 5. Project Management - JIRA

This entire project can be managed using Jira(project tracking tool), breaking it down into 2 week sprints. The first week focuses on documentation and solution design, while the second week is dedicated to implementation and testing. Each use case and test scenario is tracked as a Jira user story or task, allowing us to monitor progress and manage resources effectively.

I have created a scrum project in JIRA - Healthcare Analysis with a 2 week sprint. Created a major Epic as - Healthcare Data Analysis. Then user stories and tasks are added under each epic such as Requirement, Design Document, Data Cleaning, Analysis and so on. Jira's board view to track progress

The screenshot shows a Jira Software interface for a project titled "Health Care Data Pipeline Implementation". The current sprint is "Sprint 1- Healthcare Analysis". The board is divided into four columns: TO DO, IN PROGRESS, DONE, and TESTING. The DONE column contains 4 items, and the TESTING column contains 3 items. Each item is represented by a card with a title, a checkmark icon, a unique ID (e.g., HCDP-10, HCDP-7, HCDP-6, etc.), and a person icon. The TESTING column also includes a "HEALTHCARE DATA ANALYSIS" label. The sidebar on the left provides navigation links for PLANNING (Timeline, Backlog), DEVELOPMENT (Code), and other project management features like Project pages and Add shortcut.

## 6. Github

Throughout this process, I have maintained comprehensive documentation, including our requirement specifications, solution design documents, and database schema designs. All our PySpark code for data cleansing and result generation is version-controlled and stored in a GitHub repository. This ensures transparency, reproducibility, and ease of collaboration for our team.

<https://github.com/eshrestha21/Health-Care-insurance-Company-Analysis>

<https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/1166544219295246/2435706089467460/8156463845699645/latest.html>

## 7. Dataset Creation

Here, the data resides in AWS S3. I am leveraging the power of Databricks to ingest this data efficiently. Using PySpark, I have established a secure connection to our S3 bucket and read the various datasets - patients, subscribers, claims, and group\_subgroup - into Databricks DataFrames. This step sets the foundation for our entire data pipeline.

## Environment: Databricks, PySpark, AWS S3

### # Data stored in AWS S3

Amazon S3 > Buckets > health-insurance-capstone

**Objects (3) Info**

Name	Type	Last modified	Size	Storage class
clean-data/	Folder	-	-	-
query-result/	Folder	-	-	-
raw-data/	Folder	-	-	-

Amazon S3 > Buckets > health-insurance-capstone > raw-data/

**Objects (8) Info**

Name	Type	Last modified	Size	Storage class
claims.json	json	July 10, 2024, 16:35:27 (UTC-05:00)	13.1 KB	Standard
disease.csv	csv	July 10, 2024, 16:35:27 (UTC-05:00)	1.5 KB	Standard
group.csv	csv	July 10, 2024, 16:35:27 (UTC-05:00)	4.6 KB	Standard
grpsubgrp.csv	csv	July 10, 2024, 16:35:28 (UTC-05:00)	473.0 B	Standard
hospital.csv	csv	July 10, 2024, 16:35:28 (UTC-05:00)	1.3 KB	Standard
Patient_records.csv	csv	July 10, 2024, 16:35:29 (UTC-05:00)	5.0 KB	Standard
subgroup.csv	csv	July 10, 2024, 16:35:29 (UTC-05:00)	265.0 B	Standard
subscriber.csv	csv	July 10, 2024, 16:35:29 (UTC-05:00)	11.7 KB	Standard

### # Accessing the AWS Bucket

▶ ✓ 02:21 PM (<1s) 3

```
#Import libraries
from pyspark.sql import SparkSession, Row
from pyspark.sql.functions import col, count, avg, when, year, datediff, current_date, avg, sum, count
from pyspark.sql.types import IntegerType

# Initialize Spark session
spark = SparkSession.builder \
    .appName("Health Insurance Analysis") \
    .getOrCreate()
```

▶ ✓ 02:21 PM (<1s) 4

```
# Accessing the AWS Bucket
access_key = 'XXXXXXXXXXXX'
secret_key = 'XXXXXX+xIP9o/I'
sc._jsc.hadoopConfiguration().set("fs.s3a.access.key", access_key)
sc._jsc.hadoopConfiguration().set("fs.s3a.secret.key", secret_key)

# If you are using Auto Loader file notification mode to load files, provide the AWS Region ID.
aws_region = "us-east-2"
sc._jsc.hadoopConfiguration().set("fs.s3a.endpoint", "s3." + aws_region + ".amazonaws.com")
```

## #ExtractLoad Data in Databricks from AWS S3

▶ ✓ 06:42 PM (10s) 6

```
# Load the data from AWS S3
claims_df = spark.read.json("s3a://health-insurance-capstone/raw-data/claims.json")
disease_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/disease.csv", header=True, inferSchema=True)
group_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/group.csv", header=True, inferSchema=True)
grpsubgrp_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/grpsubgrp.csv", header=True, inferSchema=True)
hospital_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/hospital.csv", header=True, inferSchema=True)
subgroup_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/subgroup.csv", header=True, inferSchema=True)
subscriber_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/subscriber.csv", header=True, inferSchema=True)
patient_records_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/Patient_records.csv", header=True, inferSchema=True)
```

▶ (15) Spark Jobs

```
claims_df: pyspark.sql.dataframe.DataFrame = [Claim_Or_Rejected: string, SUB_ID: string ... 6 more fields]
disease_df: pyspark.sql.dataframe.DataFrame = [SubGrpID: string, Disease_ID: integer ... 1 more field]
group_df: pyspark.sql.dataframe.DataFrame = [Country: string, premium_written: integer ... 6 more fields]
grpsubgrp_df: pyspark.sql.dataframe.DataFrame = [SubGrp_ID: string, Grp_Id: string]
hospital_df: pyspark.sql.dataframe.DataFrame = [Hospital_id: string, Hospital_name: string ... 3 more fields]
subgroup_df: pyspark.sql.dataframe.DataFrame = [SubGrp_id: string, SubGrp_Name: string ... 1 more field]
subscriber_df: pyspark.sql.dataframe.DataFrame = [sub_id: string, first_name: string ... 12 more fields]
patient_records_df: pyspark.sql.dataframe.DataFrame = [Patient_id: integer, Patient_name: string ... 6 more fields]
```

▶ ✓ 06:42 PM (10s) 6

```
# EXTRACT DATA FROM AWS S3
# Reading the data from AWS S3
claims_df = spark.read.json("s3a://health-insurance-capstone/raw-data/claims.json")
disease_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/disease.csv", header=True, inferSchema=True)
group_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/group.csv", header=True, inferSchema=True)
grpsubgrp_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/grpsubgrp.csv", header=True, inferSchema=True)
hospital_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/hospital.csv", header=True, inferSchema=True)
subgroup_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/subgroup.csv", header=True, inferSchema=True)
subscriber_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/subscriber.csv", header=True, inferSchema=True)
patient_records_df = spark.read.csv("s3a://health-insurance-capstone/raw-data/Patient_records.csv", header=True, inferSchema=True)
```

▶ (15) Spark Jobs

```
claims_df: pyspark.sql.dataframe.DataFrame
  Claim_Or_Rejected: string
  SUB_ID: string
  claim_amount: string
  claim_date: string
  claim_id: long
  claim_type: string
  disease_name: string
  patient_id: long

disease_df: pyspark.sql.dataframe.DataFrame
  SubGrpID: string
  Disease_ID: integer
  Disease_name: string

group_df: pyspark.sql.dataframe.DataFrame
  Country: string
  premium_written: integer
  zipcode: integer
  Grp_Id: string
  Grp_Name: string
  Grp_Type: string
```

```

Just now (ts)
8
Python

# Print schema and display data for each DataFrame
print("Schema of claims_df:")
claims_df.printSchema()
claims_df.show()

> (1) Spark Jobs

Schema of claims_df:
root
|-- Claim_Or_Rejected: string (nullable = true)
|-- SUB_ID: string (nullable = true)
|-- claim_amount: string (nullable = true)
|-- claim_date: string (nullable = true)
|-- claim_id: long (nullable = true)
|-- claim_type: string (nullable = true)
|-- disease_name: string (nullable = true)
|-- patient_id: long (nullable = true)

+-----+-----+-----+-----+-----+-----+
|Claim_Or_Rejected|SUB_ID|claim_amount|claim_date|claim_id|claim_type|disease_name|patient_id|
+-----+-----+-----+-----+-----+-----+
|          N|SUBID1000|    79874|1949-03-14|      0| claims of value|Galactosemia| 187158|
|          N|SUBID10001|   151142|1970-03-16|      1| claims of policy|Bladder cancer| 112766|
|          N|SUBID10002|    59924|2008-02-03|      2| claims of value|Kidney cancer| 199252|
|          N|SUBID10003|   143120|1995-02-08|      3| claims of fact| Suicide| 133424|
|          Y|SUBID10004|   168634|1967-05-23|      4| claims of value|Food allergy| 172579|
|          N|SUBID10005|    64840|1991-10-04|      5| claims of policy| Whiplash| 171328|
|          N|SUBID10006|   26880|1991-03-26|      6| claims of fact| Sunbathing| 107794|

```

```

Just now (<1s)
8
Python

# Print schema and display data for each DataFrame
print("Schema of disease_df:")
disease_df.printSchema()
disease_df.show()

> (1) Spark Jobs

Schema of disease_df:
root
|-- SubGrpID: string (nullable = true)
|-- Disease_ID: integer (nullable = true)
|-- Disease_name: string (nullable = true)

+-----+-----+
|SubGrpID|Disease_ID|Disease_name|
+-----+-----+
|     S101|    110001|    Beriberi|
|     S101|    110002|      Scurvy|
|     S101|    110003|      Goitre|
|     S101|    110004| Osteoporosis|
|     S101|    110005|      Rickets|
|     S101|    110006|      Anemia|
|     S102|    110007|      Fractures|
|     S102|    110008| Heart Attack|
|     S102|    110009|       Burns|
|     S102|    110010|      Choking|
|     S102|    110011|       Stroke|

```

Just now (<1s) 8

```
# Print schema and display data for each DataFrame
print("Schema of group_df:")
group_df.printSchema()
group_df.show()

▶ (1) Spark Jobs
```

Schema of group\_df:

root

```
|-- Country: string (nullable = true)
|-- premium_written: integer (nullable = true)
|-- zipcode: integer (nullable = true)
|-- Grp_Id: string (nullable = true)
|-- Grp_Name: string (nullable = true)
|-- Grp_Type: string (nullable = true)
|-- city: string (nullable = true)
|-- year: integer (nullable = true)
```

Country	premium_written	zipcode	Grp_Id	Grp_Name	Grp_Type	city	year
India	72000	482018	GRP101	Life Insurance Co...	Govt.	Mumbai	1956
India	45000	482049	GRP102	HDFC Standard Lif...	Private	Mumbai	2000
India	64000	482030	GRP103	Max Life Insuranc...	Private	Delhi	2000
India	59000	482028	GRP104	ICICI Prudential ...	Private	Mumbai	2000
India	37000	482014	GRP105	Kotak Mahindra Li...	Private	Mumbai	2001
India	89000	482011	GRP106	Aditya Birla Sun ...	Private	Mumbai	2000
India	70000	482006	GRP107	TATA AIG Life Ins...	Private	Mumbai	2001

Just now (<1s)

```
print("Schema of hospital_df:")
hospital_df.printSchema()
hospital_df.show()

▶ (1) Spark Jobs
```

Schema of hospital\_df:

root

```
|-- Hospital_id: string (nullable = true)
|-- Hospital_name: string (nullable = true)
|-- city: string (nullable = true)
|-- state: string (nullable = true)
|-- country: string (nullable = true)
```

Hospital_id	Hospital_name	city	state	country
H1000	All India Institu...	New Delhi	Nan	India
H1001	Medanta The Medicity	Gurgaon	Haryana	India
H1002	The Christian Med...	Vellore	Tamil Nadu	India
H1003	PGIMER - Postgrad...	Chandigarh	Haryana	India
H1004	Apollo Hospital -...	Chennai	Tamil Nadu	India
H1005	P. D. Hindujas Nat...	Mumbai	Maharashtra	India
H1006	Breach Candy Hosp...	Mumbai	Maharashtra	India
H1007	Fortis Flt. Lt. R...	New Delhi	Nan	India
H1008	King Edward Memor...	Mumbai	Maharashtra	India
H1009	Indraprastha Aopol...	Delhi	Nan	India

Just now (<1s)

```
print("Schema of subgroup_df:")
subgroup_df.printSchema()
subgroup_df.show()
```

(1) Spark Jobs

Schema of subgroup\_df:

```
root
|--- SubGrp_id: string (nullable = true)
|--- SubGrp_Name: string (nullable = true)
|--- Monthly_Premium: integer (nullable = true)

+-----+
|SubGrp_id| SubGrp_Name|Monthly_Premium|
+-----+
| S101|Deficiency Diseases| 3000|
| S102| Accident| 1000|
| S103| Physiology| 2000|
| S104| Therapy| 1500|
| S105| Allergies| 2300|
| S106| Self inflicted| 1200|
| S107| Cancer| 3200|
| S108| Infectious disease| 1500|
| S109| Hereditary| 2000|
| S110| Viral| 1000|
+-----+
```

Just now (1s)

```
print("Schema of subscriber_df:")
subscriber_df.printSchema()
subscriber_df.show()
```

(1) Spark Jobs

Schema of subscriber\_df:

```
|-- birth_date: date (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Phone: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- City: string (nullable = true)
 |-- Zip Code: integer (nullable = true)
 |-- Subgrp_id: string (nullable = true)
 |-- Elig_ind: string (nullable = true)
 |-- eff_date: date (nullable = true)
 |-- term_date: date (nullable = true)

+-----+
| sub_id|first_name| last_name| Street|Birth_date|Gender| Phone|Country| City|Zip Code|Subgrp_id|Elig_ind| eff_date| term_date|
+-----+
|SUBID10000| Harbir|Vishwakarma| Baria Marg|1924-06-30|Female|+91 0112009318| India| Rourkela| 767058| S107| Y|194-06-30|1954-01-14|
|SUBID10001| Brahmdev| Sonkar| Lala Marg|1948-12-20|Female|+91 1727749552| India| Tiruvottiyur| 34639| S105| Y|198-12-20|1970-05-16|
|SUBID10002| Ujjwal| Ballari| Manmohan Marg|1960-04-15|Male|+91 9876543210| India| Dehradoon| 783001| S106| Y|196-04-15|1980-04-15|
|SUBID10003| Devnath| Atasi| Tandojam Marg|1967-10-02|Male|+91 9747336855| India| Panvel| 410041| S108| Y|1967-10-02|1987-06-06|
|SUBID10004| Manish| Male| Sunbathinol|1967-06-06|Male|+91 4354294043| India| Mumbai| 400011| S109| Y|1967-06-06|1987-06-06|
```

Just now (<1s)

```
print("Schema of patient_records_df:")
patient_records_df.printSchema()
patient_records_df.show()
```

(1) Spark Jobs

Schema of patient\_records\_df:

```
root
|--- Patient_id: integer (nullable = true)
|--- Patient_name: string (nullable = true)
|--- patient_gender: string (nullable = true)
|--- patient_birth_date: date (nullable = true)
|--- patient_phone: string (nullable = true)
|--- disease_name: string (nullable = true)
|--- city: string (nullable = true)
|--- hospital_id: string (nullable = true)

+-----+
|Patient_id|Patient_name|patient_gender|patient_birth_date| patient_phone| disease_name| city|hospital_id|
+-----+
| 187158| Harbir| Female| 1924-06-30|+91 0112009318| Galactosemia| Rourkela| H1001|
| 112766| Brahmdev| Female| 1948-12-20|+91 1727749552| Bladder cancer| Tiruvottiyur| H1016|
| 199252| Ujjwal| Male| 1980-04-16|+91 8547451606| Kidney cancer| Berhampur| H1009|
| 133424| Ballari| Female| 1969-09-25|+91 0106026841| Suicide| Bihar Sharif| H1017|
| 172579| Devnath| Female| 1946-05-01|+91 1868774631| Food allergy| Bidhannagar| H1019|
| 171320| Atasi| Male| 1967-10-02|+91 9747336855| Whiplash| Amravati| H1013|
| 1077941| Manish| Male| 1967-06-06|+91 4354294043| Sunbathinol| Panvel| H1004|
```

## 8. Data Cleaning

Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct. There is no one absolute way to prescribe the exact steps in the data cleaning process because the processes will vary from dataset to dataset.

- Cleaning Activity

- First check if there are null values in dataset
- Count the total Null values for each column
- And then replace the null values for specific columns by NA
- Check the If three are duplicates records
- If there are duplicates then drop duplicates

```
# TRANSFORM DATA

# Function to clean a DataFrame
def clean_data(df, df_name):
    # 1. Replace empty strings with None (null)
    df = df.replace('', None)

    # 2. Check for null values and count them for each column
    null_counts = df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns])
    null_counts.show()

    # 3. Replace remaining null values with 'NA' for specific columns (customize as needed)
    columns_to_replace = df.columns
    df = df.fillna("NA", subset=columns_to_replace)

    # 4. Check for duplicates and drop duplicates
    initial_count = df.count()
    df = df.dropDuplicates()
    final_count = df.count()

    print(f"Number of duplicates dropped in {df_name}: {initial_count - final_count}")

return df
```

07:01 PM (14s) 12

```
# Clean each DataFrame using user defined function clean_data()
claims_df_cleaned = clean_data(claims_df, "claims_df")
disease_df_cleaned = clean_data(disease_df, "disease_df")
group_df_cleaned = clean_data(group_df, "group_df")
grpsubgrp_df_cleaned = clean_data(grpsubgrp_df, "grpsubgrp_df")
hospital_df_cleaned = clean_data(hospital_df, "hospital_df")
subgroup_df_cleaned = clean_data(subgroup_df, "subgroup_df")
subscriber_df_cleaned = clean_data(subscriber_df, "subscriber_df")
patient_records_df_cleaned = clean_data(patient_records_df, "patient_records_df")
```

▶ (56) Spark Jobs

- ▶ claims\_df\_cleaned: pyspark.sql.dataframe.DataFrame = [Claim\_Or\_Rejected: string, SUB\_ID: string ... 6 more fields]
- ▶ disease\_df\_cleaned: pyspark.sql.dataframe.DataFrame = [SubGrpID: string, Disease\_ID: integer ... 1 more field]
- ▶ group\_df\_cleaned: pyspark.sql.dataframe.DataFrame = [Country: string, premium\_written: integer ... 6 more fields]
- ▶ grpsubgrp\_df\_cleaned: pyspark.sql.dataframe.DataFrame = [SubGrp\_ID: string, Grp\_Id: string]
- ▶ hospital\_df\_cleaned: pyspark.sql.dataframe.DataFrame = [Hospital\_Id: string, Hospital\_name: string ... 3 more fields]
- ▶ subgroup\_df\_cleaned: pyspark.sql.dataframe.DataFrame = [SubGrp\_id: string, SubGrp\_Name: string ... 1 more field]
- ▶ subscriber\_df\_cleaned: pyspark.sql.dataframe.DataFrame = [sub\_id: string, first\_name: string ... 12 more fields]
- ▶ patient\_records\_df\_cleaned: pyspark.sql.dataframe.DataFrame = [Patient\_id: integer, Patient\_name: string ... 6 more fields]

Claim_Or_Rejected	SUB_ID	claim_amount	claim_date	claim_id	claim_type	disease_name	patient_id
0	0	0	0	0	0	0	0

Number of duplicates dropped in claims\_df: 0

SubGrpID	Disease_ID	Disease_name
0	0	0

Number of duplicates dropped in disease\_df: 0

Country	premium_written	zipcode	Grp_Id	Grp_Name	Grp_Type	city	year
0	0	0	0	0	0	0	0

Number of duplicates dropped in group\_df: 0

SubGrp_ID	Grp_Id
0	0

Number of duplicates dropped in grpsubgrp\_df: 0

Hospital_id	Hospital_name	city	state	country
0	0	0	0	0

```
▶ [ ] subgroup_df_cleaned: pyspark.sql.dataframe.DataFrame = [Subgrp_id: string, SubGrp_Name: string ... 1 more field]
▶ [ ] subscriber_df_cleaned: pyspark.sql.dataframe.DataFrame = [sub_id: string, first_name: string ... 12 more fields]
▶ [ ] patient_records_df_cleaned: pyspark.sql.dataframe.DataFrame = [Patient_id: integer, Patient_name: string ... 6 more fields]
+-----+-----+-----+
|SubGrp_id|SubGrp_Name|Monthly_Premium|
+-----+-----+-----+
|      0|          0|            0|
+-----+-----+-----+
Number of duplicates dropped in subgroup_df: 0
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|sub_id|first_name|last_name|Street|Birth_date|Gender|Phone|Country|City|Zip Code|Subgrp_id|Elig_ind|eff_date|term_date|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|      0|       27|        0|     0|       0|     0|    3|     0|     0|       0|       2|      4|     0|     0|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
Number of duplicates dropped in subscriber_df: 0
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Patient_id|Patient_name|patient_gender|patient_birth_date|patient_phone|disease_name|city|hospital_id|
+-----+-----+-----+-----+-----+-----+-----+-----+
|      0|       17|        0|           0|        2|        0|     0|       0|
+-----+-----+-----+-----+-----+-----+-----+-----+
Number of duplicates dropped in patient_records_df: 0
```

## Table: Patient Before Cleaning

▶ ▾ ✓ 02:37 PM (1s)

patient\_records\_df.show()

▶ (1) Spark Jobs

187158   Harbir   Female   1924-06-30 +91 0112009318   Galactosemia   Rourkela   H1001
112766   Brahmdev   Female   1948-12-20 +91 1727749552   Bladder cancer   Tiruvottiyur   H1016
199252   Ujjawal   Male   1980-04-16 +91 8547451606   Kidney cancer   Berhampur   H1009
133424   Ballari   Female   1969-09-25 +91 0106026841   Suicide   Bihar Sharif   H1017
172579   Devnath   Female   1946-05-01 +91 1868774631   Food allergy   Bidhannagar   H1019
171320   Atasi   Male   1967-10-02 +91 9747336855   Whiplash   Amravati   H1013
107794   Manish   Male   1967-06-06 +91 4354294043   Sunbathing   Panvel   H1004
130339   Aakar   Female   1925-03-05 +91 2777633911   Drug consumption   Bihar Sharif   H1000
110377   Gurudas   Male   1945-05-06 +91 1232859381   Dengue   Kamarhati   H1001
149367   null   Male   1925-06-12 +91 1780763280   Head banging   Bangalore   H1013
156168   null   Male   1976-02-03 +91 5586075345   Fanconi anaemia   Rajkot   H1004
114241   null   Female   1955-01-22 +91 4146391938   Breast cancer   Ghaziabad   H1015
146382   Dharmadaas   Male   1964-04-29 +91 6345482027   Anthrax   Bhalswa Jahangir Pur   H1019
132748   Brahmvir   Male   1991-11-11 +91 7316972612   Cystic fibrosis   Ambala   H1018
167340   null   Female   1981-01-25 +91 2960004518   Galactosemia   Surendranagar Dud...   H1003
135184   Bhagvan   Female   1966-07-24 +91 0297693485   Dengue   Bhimavaram   H1018
179662   Amritkala   Female   1933-11-20 +91 0537157280   Smallpox   Meerut   H1018
184479   Bandhu   Male   1996-10-15 +91 0695289163   Pollen allergy   Chinsurah   H1010
156988   Bhagavaana   Female   1935-09-16 +91 6071745855   Breast cancer   Shahjahanpur   H1012
132870   null   Female   1924-11-09 +91 8906694405   Glaucoma   Jabalpur   H1017

## After Cleaning

▶ ▾ ✓ 02:37 PM (1s)

patient\_records\_df\_cleaned.show()

▶ (2) Spark Jobs

132748   Brahmvir   Male   1991-11-11 +91 7316972612   Cystic fibrosis   Ambala   H1018
172579   Devnath   Female   1946-05-01 +91 1868774631   Food allergy   Bidhannagar   H1019
146382   Dharmadaas   Male   1964-04-29 +91 6345482027   Anthrax   Bhalswa Jahangir Pur   H1019
114241   NA   Female   1955-01-22 +91 4146391938   Breast cancer   Ghaziabad   H1015
148137   Umang   Female   1963-07-14 +91 948538770   Pet allergy   Haridwar   H1002
171320   Atasi   Male   1967-10-02 +91 9747336855   Whiplash   Amravati   H1013
187158   Harbir   Female   1924-06-30 +91 0112009318   Galactosemia   Rourkela   H1001
199252   Ujjawal   Male   1980-04-16 +91 8547451606   Kidney cancer   Berhampur   H1009
149367   NA   Male   1925-06-12 +91 1780763280   Head banging   Bangalore   H1013
133424   Ballari   Female   1969-09-25 +91 0106026841   Suicide   Bihar Sharif   H1017
130339   Aakar   Female   1925-03-05 +91 2777633911   Drug consumption   Bihar Sharif   H1000
156988   Bhagavaana   Female   1935-09-16 +91 6071745855   Breast cancer   Shahjahanpur   H1012
112766   Brahmdev   Female   1948-12-20 +91 1727749552   Bladder cancer   Tiruvottiyur   H1016
107794   Manish   Male   1967-06-06 +91 4354294043   Sunbathing   Panvel   H1004
110377   Gurudas   Male   1945-05-06 +91 1232859381   Dengue   Kamarhati   H1001
132870   NA   Female   1924-11-09 +91 8906694405   Glaucoma   Jabalpur   H1017
179662   Amritkala   Female   1933-11-20 +91 0537157280   Smallpox   Meerut   H1018
184479   Bandhu   Male   1996-10-15 +91 0695289163   Pollen allergy   Chinsurah   H1010

only showing top 20 rows

**Table: Subscriber**  
Before Cleaning

02:24 PM (1s) 12 Python

```
subscriber_df.show()

(1) Spark Jobs
+-----+
|SUBID1004| Devnath| Srivastav| Magar Zila|1946-05-01|Female|+91 1868774631| India| Bidhannagar| 531742| S10| N|196
|SUBID1005| Atasi| Seth| Khatri Nagar|1967-10-02| Male|+91 9747336855| India| Amravati| 229062| S104| Y|198
|SUBID1006| Manish| Maurya|Swaminathan Chowk|1967-06-06| Male|+91 4354294043| India| Panvel| 438733| S109| null|198
|SUBID1007| Aakar| Yadav| Swamy|1925-03-05|Female|+91 2777633911| India| Bihar Sharif| 535907| S104| N|194
|SUBID1008| Gurudas| Gupta| Sarin Nagar|1945-05-06| Male|+91 1232859381| India| Kamarhati| 933226| S103| Y|196
|SUBID1009| null| Gupta| Thakur Circle|1925-06-12| Male|+91 1780763280| India| Bangalore| 957469| S105| Y|194
|SUBID1010| null| Divedi| Dhillon|1976-02-03| Male|+91 5586075345| India| Rajkot| 911319| S102| Y|199
|SUBID1011| null|Vishwakarma| Rajagopalan|1955-01-22|Female|+91 4146391938| India| Ghaziabad| 337042| S106| N|197
|SUBID1012|Dharmadaas| Tiwari| Rama|1964-04-29| Male|+91 6345482027| India|Bhalswa Jahangir Pur| 430793| S103| N|198
|SUBID1013| Brahmvir| Rai| Shah Path|1991-11-11| Male|+91 7316972612| India| Ambala| 249898| S106| N|201
1-11-11|2020-05-23|
+-----+
```

## After Cleaning

4 minutes ago (1s) 13 Python

```
subscriber_df_cleaned.show();

(2) Spark Jobs
+-----+
|SUBID10013| Brahmvir| Rai| Shah Path|1991-11-11| Male|+91 7316972612| India| Ambala| 249898| S106| N|201
1-11-11|2020-05-23|
|SUBID10012|Dharmadaas| Tiwari| Rama|1964-04-29| Male|+91 6345482027| India|Bhalswa Jahangir Pur| 430793| S103| N|198
4-04-29|1988-02-07|
|SUBID10011| NA|Vishwakarma| Rajagopalan|1955-01-22|Female|+91 4146391938| India| Ghaziabad| 337042| S106| N|197
5-01-22|1978-11-02|
|SUBID10018|Bhagavaana| Kumar| Kulkarni Zila|1935-09-16|Female|+91 6071745855| India| Shahjahanpur| 597276| S101| N|195
5-09-16|1958-05-31|
|SUBID10002| Ujjawal| Devi| Mammen Zila|1980-04-16| Male|+91 8547451606| India| Berhampur| 914455| S106| N|200
0-04-16|2008-05-04|
|SUBID10009| NA| Gupta| Thakur Circle|1925-06-12| Male|+91 1780763280| India| Bangalore| 957469| S105| Y|194
5-06-12|1953-08-30|
|SUBID1020| Umang| Srivastav| Balay Chowk|1963-07-14|Female|+91 9485838770| India| Haridwar| 181692| S109| Y|198
3-07-14|1986-01-15|
|SUBID1010| NA| Divedi| Dhillon|1976-02-03| Male|+91 5586075345| India| Rajkot| 911319| S102| Y|199
6-02-03|2002-01-27|
|SUBID10000| Harbir|Vishwakarma| Baria Marg|1924-06-30|Female|+91 0112009318| India| Rourkela| 767058| S107| Y|194
4-06-30|1954-01-14|
|SUBID10014| NAI| Srivastav| Chandra Path|1981-01-25|Female|+91 29600045181| India|Surendranagar_Dud...| 111966| S102| N|200
+-----+
```

## 9. Writing the cleaned data to AWS S3.

After cleaning and analysis, we now have high-quality, processed/cleaned data which is exported back to our AWS S3 bucket, creating a new 'cleaned-data' folder. This helps to ensure that we maintain a clear separation between raw and processed data, and provides a clean dataset for future use or additional analytics.

Environment: Databricks, PySpark, AWS S3

```
# WRITING DATA
```

1 minute ago (27s) 14 Python

```
# Writing the cleaned data to AWS S3
claims_df_cleaned.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/clean-data/claims")
disease_df_cleaned.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/clean-data/disease")
group_df_cleaned.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/clean-data/group")
grpsubgrp_df_cleaned.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/clean-data/grpsubgrp")
hospital_df_cleaned.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/clean-data/hospital")
subgroup_df_cleaned.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/clean-data/subgroup")
subscriber_df_cleaned.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/clean-data/subscriber")
patient_records_df_cleaned.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/clean-data/patient")
```

(14) Spark Jobs

- Job 259 [View](#) (Stages: 1/1)
- Job 260 [View](#) (Stages: 1/1)
- Job 261 [View](#) (Stages: 1/1, 1 skipped)
- Job 262 [View](#) (Stages: 1/1)
- Job 263 [View](#) (Stages: 1/1, 1 skipped)
- Job 264 [View](#) (Stages: 1/1)
- Job 265 [View](#) (Stages: 1/1, 1 skipped)
- Job 266 [View](#) (Stages: 1/1)
- Job 267 [View](#) (Stages: 1/1, 1 skipped)
- Job 268 [View](#) (Stages: 1/1)
- Job 269 [View](#) (Stages: 1/1, 1 skipped)
- Job 270 [View](#) (Stages: 1/1)
- Job 271 [View](#) (Stages: 1/1, 1 skipped)
- Job 272 [View](#) (Stages: 1/1)

[Amazon S3](#)

Buckets

- Access Grants
- Access Points
- Object Lambda Access Points
- Multi-Region Access Points
- Batch Operations
- IAM Access Analyzer for S3

Block Public Access settings for this account

▼ Storage Lens

- Dashboards
- Storage Lens groups
- AWS Organizations settings

Feature spotlight: [AWS Lambda](#)

► AWS Marketplace for S3

[Amazon S3 > Buckets > health-insurance-capstone > clean-data/](#)

clean-data/

Objects (0) [Info](#) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) Actions Create folder [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix:

Name	Type	Last modified	Size	Storage class
claims/	Folder	-	-	-
disease/	Folder	-	-	-
group/	Folder	-	-	-
grpsubgrp/	Folder	-	-	-
hospital/	Folder	-	-	-
patient/	Folder	-	-	-
subgroup/	Folder	-	-	-
subscriber/	Folder	-	-	-

[Amazon S3](#)

Buckets

- Access Grants
- Access Points
- Object Lambda Access Points
- Multi-Region Access Points
- Batch Operations
- IAM Access Analyzer for S3

Block Public Access settings for this account

▼ Storage Lens

- Dashboards
- Storage Lens groups
- AWS Organizations settings

Feature spotlight: [AWS Lambda](#)

► AWS Marketplace for S3

[Amazon S3 > Buckets > health-insurance-capstone > clean-data/ > claims/](#)

claims/

Objects (6) [Info](#) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) Actions Create folder [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

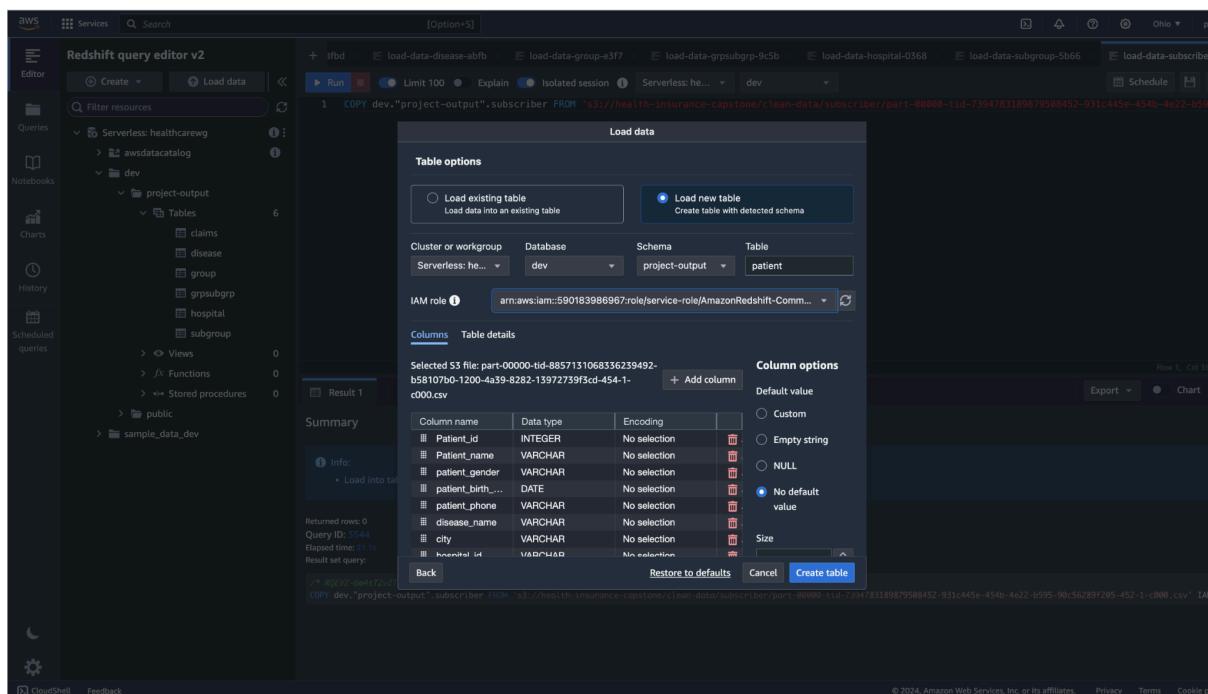
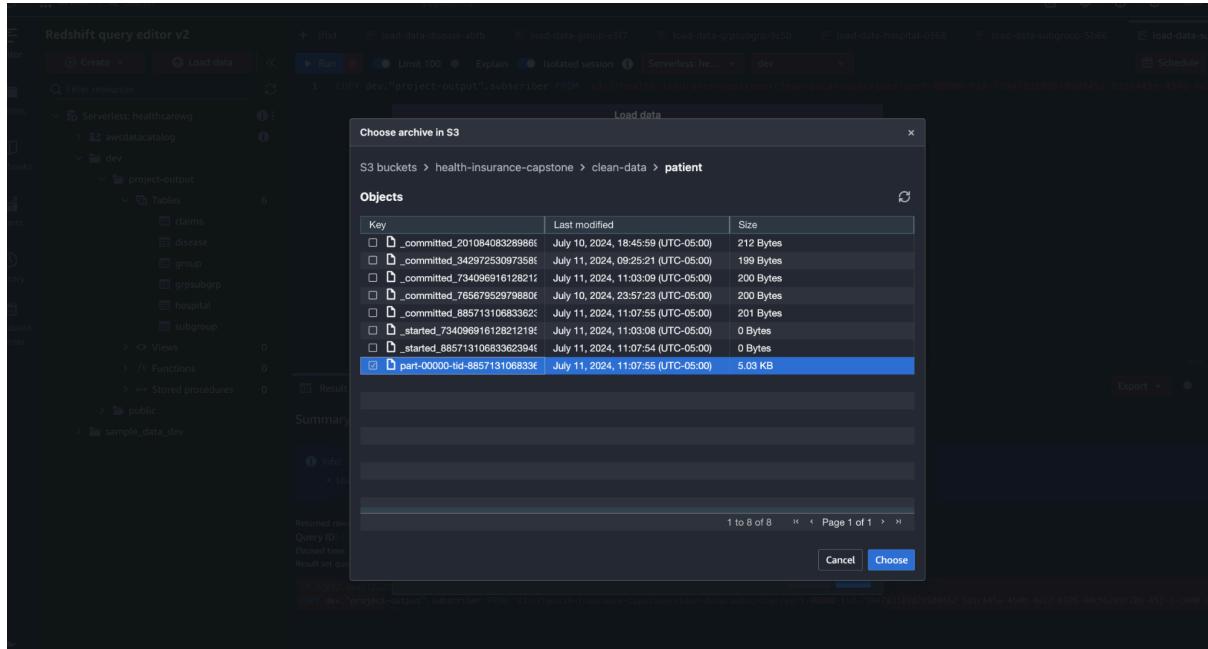
Find objects by prefix:

Name	Type	Last modified	Size	Storage class
.Committed_2398982369505752359	-	July 10, 2024, 18:35:32 (UTC-05:00)	115.0 B	Standard
.committed_5229213139375621544	-	July 10, 2024, 18:45:40 (UTC-05:00)	212.0 B	Standard
.started_2398982369505752359	-	July 10, 2024, 18:35:31 (UTC-05:00)	0 B	Standard
.started_5229213139375621544	-	July 10, 2024, 18:45:39 (UTC-05:00)	0 B	Standard
.SUCCESS	-	July 10, 2024, 18:45:40 (UTC-05:00)	0 B	Standard
part-00000-id-5229213139375621544-2380aa3-1-2000.csv	csv	July 10, 2024, 18:45:39 (UTC-05:00)	4.7 KB	Standard

## 10. Upload cleaned data corresponding to each data set into a redshift table.

For efficient querying and long-term storage, the cleaned data is loaded to S3 to AWS Redshift. Each dataset is loaded into its corresponding table in the Redshift cluster. Similarly, each use case output is stored in a separate Redshift table within a schema named **Project-Output**.

**Environment:** AWS Redshift, AWS S3



SMS Services Search [Option+S]

Redshift query editor v2

Editor Queries Notebooks Charts History Scheduled queries

Services Search [Option+S]

Redshift query editor v2

Load data

Review and load data

Serverless: healthcarewg dev project-output patient

IAM role: arn:aws:iam::590183986967:role/service-role/AmazonRedshift-CommandsAccessRole-20240711T115530

Table summary

Column name	Data type	Encoding
Patient_id	INTEGER	No selection
Patient_name	VARCHAR	No selection
patient_gender	VARCHAR	No selection
patient_birth_date	DATE	No selection
patient_phone	VARCHAR	No selection
disease_name	VARCHAR	No selection
city	VARCHAR	No selection
hospital_id	VARCHAR	No selection

Summary

Info: + Load into table

Returned rows: 0 Query ID: 3544 Elapsed time: 21.1s Result set query:

```
/* RQEIV2-vm4HtzXN */  
COPY dev."project-output".subscriber FROM 's3://health-insurance-capstone/clean-data/subscriber/part-00000-tid-7394783189879588452-931c445e-454b-4e22-b595-98c56289f205-452-1-c000.csv' TAM_ROLE
```

Back Cancel Load data

Row 1, Col 1

Export Chart

load-data-disease load-data-hospital-0368 load-data-subgroup-5b66 load-data-subscriber

Ohio

[Option+S]

SMS Services Search [Option+S]

Redshift query editor v2

Editor Queries Notebooks Charts History Scheduled queries

Services Search [Option+S]

Redshift query editor v2

Load data

Review and load data

Serverless: healthcarewg dev project-output patient

Table summary

Column name	Data type	Encoding
Patient_id	INTEGER	No selection
Patient_name	VARCHAR	No selection
patient_gender	VARCHAR	No selection
patient_birth_date	DATE	No selection
patient_phone	VARCHAR	No selection
disease_name	VARCHAR	No selection
city	VARCHAR	No selection
hospital_id	VARCHAR	No selection

Summary

Info: + Load into table

Returned rows: 0 Query ID: 3601 Elapsed time: 14.6s Result set query:

```
/* RQEIV2-vm4HtzXN */  
COPY dev."project-output".patient FROM 's3://health-insurance-capstone/clean-data/patient/part-00000-tid-8857131068336239492-b58107b0-1200-4a39-8282-13972739F3cd-454-1-c000.csv' TAM_ROLE
```

Back Cancel Load data

Row 1, Col 33

Export Chart

load-data-group-e3f7 load-data-grpsubgrp-95b load-data-hospital-0368 load-data-subgroup-5b66 load-data-subscriber-d1b4 load-data-patient

Ohio

[Option+S]

## 11. Use Cases

- **Disease Analysis:** Identify the disease with the maximum number of claims.

Here, insurance claims are grouped by disease name, count the occurrences, and identify the disease with the maximum claims. The resulting DataFrame is displayed and then saved as a permanent table in Databricks, allowing for persistent storage and future analysis.

```
# 1. Disease with maximum number of claim

from pyspark.sql.functions import col

# Group by disease_name and count the occurrences, then get the one with the maximum count
max_claims_disease_df = claims_df_cleaned.groupBy("disease_name") \
    .count() \
    .orderBy(col("count").desc()) \
    .limit(1)

# Display the result
max_claims_disease_df.show()

# Saving as a permanent table
permanent_table_name = "max_claims_disease"
max_claims_disease_df.write.mode("overwrite").saveAsTable(permanent_table_name)

# Saving data to AWS S3
max_claims_disease_df.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/use-case-output/max_claims_disease")

(15) Spark Jobs
▶ max_claims_disease_df: pyspark.sql.dataframe.DataFrame = [disease_name: string, count: long]
+-----+
|disease_name|count|
+-----+
| Pet allergy|   3|
+-----+
```

- **Subscriber Analysis:** Find subscribers under the age of 30 who subscribe to any subgroup.

This query joins subscriber and subgroup data, filters out subscribers under 30 years old, and selects all relevant subscriber columns

```
# 2. Subscribers under age 30 subscribing to any subgroup
from pyspark.sql.functions import datediff, current_date

subscribers_under_30_df = subscriber_df.join(subgroup_df, subscriber_df["Subgrp_id"] == subgroup_df["SubGrp_id"]) \
    .filter(datediff(current_date(), col("Birth_date")) / 365 < 30) \
    .select(subscriber_df["*"])

subscribers_under_30_df.show()

#Saving as a permanent table
permanent_table_name = "subscribers_under_30_df"
subscribers_under_30_df.write.format("csv").mode("overwrite").saveAsTable(permanent_table_name)
# Saving data to AWS S3
subscribers_under_30_df.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/use-case-output/subscribers_under_30")

▶ (4) Spark Jobs
▶ [subscribers_under_30_df: pyspark.sql.dataframe.DataFrame = [sub_id: string, first_name: string ... 12 more fields]
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| sub_id | first_name | last_name | Street | Birth_date | Gender | Phone | Country | City | Zip Code | Subgrp_id | Elig_ind | eff_date | t
erm_date |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|SUBID10017| Bandhu | Seth | Varughese | 1996-10-15 | Male | +91 0695289163 | India | Chinsurah | 136713 | S108 | N | 2016-10-15 | 2018-06-08 |
|SUBID10083| Bhilangana | Pandit | Ramachandran Path | 1995-01-04 | Female | +91 6653069630 | India | Fatehpur | 359466 | S109 | Y | 2015-01-04 | 2017-10-05 |
|SUBID10093| Chandavarman | Singh | Sarkar Circle | 1997-05-10 | Others | +91 6559031791 | India | Navi Mumbai | 83240 | S110 | N | 2017-05-10 | 2022-08-27 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

- **Group Analysis:** Determine which group has the maximum number of subgroups.

This query groups data by Grp\_Id and counts the subgroups for each group. It then identifies the group with the most subgroups, displays the result, and saves this information as a permanent table in Databricks, ensuring the data is persistently stored for future analysis

```
# 3. Group with maximum subgroup
# Group by Grp_Id and count the subgroups, then get the one with the maximum count
max_subgroup_group_df = grp_subgrp_df.groupBy("Grp_Id") \
    .count() \
    .orderBy(col("count").desc()) \
    .limit(1)

# Display the result
max_subgroup_group_df.show()

# Saving as a permanent table
permanent_table_name = "max_subgroup_group"
max_subgroup_group_df.write.mode("overwrite").saveAsTable(permanent_table_name)
# Saving data to AWS S3
max_subgroup_group_df.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/use-case-output/max_subgroup_group")

▶ (13) Spark Jobs
▶ [max_subgroup_group_df: pyspark.sql.dataframe.DataFrame = [Grp_Id: string, count: long]
+-----+-----+
| Grp_Id | count |
+-----+-----+
| GRP104 | 2 |
+-----+-----+
```

- **Hospital Analysis:** Identify the hospital that serves the most number of patients.

This DataFrame groups the data by hospital\_id, counts the number of patients, and selects the hospital with the highest patient count.

```
from pyspark.sql.functions import col

# Group by hospital_id and count the patients, then get the hospital with the maximum count
most_patients_hospital_df = patient_records_df.groupBy("hospital_id") \
    .count() \
    .orderBy(col("count").desc()) \
    .limit(1)

# Display the result
most_patients_hospital_df.show()

# Saving as a permanent table
permanent_table_name = "most_patients_hospital"
most_patients_hospital_df.write.mode("overwrite").saveAsTable(permanent_table_name)
# Saving data to AWS S3
most_patients_hospital_df.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/use-case-output/most_patients_hospital")

▶ (13) Spark Jobs
▶ most_patients_hospital_df: pyspark.sql.dataframe.DataFrame = [hospital_id: string, count: long]
+-----+---+
|hospital_id|count|
+-----+---+
|      H1017|     9|
+-----+---+
```

- **Subgroup Analysis:** Find out which subgroup is subscribed to the most.

```
# 5. Subgroup subscribed most number of times
from pyspark.sql.functions import col

# Group by Subgrp_id and count the occurrences, then get the one with the maximum count
most_subscribed_subgroup_df = subscriber_df.groupBy("Subgrp_id") \
    .count() \
    .orderBy(col("count").desc()) \
    .limit(1)

# Display the result
most_subscribed_subgroup_df.show()

# Saving as a permanent table
permanent_table_name = "most_subscribed_subgroup"
most_subscribed_subgroup_df.write.mode("overwrite").saveAsTable(permanent_table_name)

# Saving data to AWS S3
most_subscribed_subgroup_df.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/use-case-output/most_subscribed_subgroup")

▶ (13) Spark Jobs
▶ most_subscribed_subgroup_df: pyspark.sql.dataframe.DataFrame = [Subgrp_id: string, count: long]
+-----+---+
|Subgrp_id|count|
+-----+---+
|      S104|    13|
+-----+---+
```

This DataFrame groups the data by Subgrp\_id, counts the subscriptions, and selects the subgroup with the highest subscription count.

- **Claims Rejection Analysis:** Calculate the total number of claims that were rejected.

The script filters claims\_df for rejected claims and counts them. To save this count as a permanent table, it converts the count into a DataFrame and saves it under total\_rejected\_claims.

```
# 6. Total number of claims which were rejected
from pyspark.sql import Row

total_rejected_claims = claims_df.filter(col("Claim_Or_Rejected") == "Rejected") \
    .count()
#print("Total number of rejected claims:", total_rejected_claims)
# Convert the count to a DataFrame
total_rejected_claims_df = spark.createDataFrame([Row(total_rejected_claims=total_rejected_claims)])

# Display the result
total_rejected_claims_df.show()

# Saving as a permanent table
permanent_table_name = "total_rejected_claims"
total_rejected_claims_df.write.mode("overwrite").saveAsTable(permanent_table_name)

# Saving data to AWS S3
total_rejected_claims_df.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/use-case-output/
total_rejected_claims")

(15) Spark Jobs
▶ total_rejected_claims_df: pyspark.sql.dataframe.DataFrame = [total_rejected_claims: long]
-----+
total_rejected_claims|
-----+
  0|
-----+
```

- **Claims Origin Analysis:** Determine the city from which most claims are coming.

This script joins claims\_df with subscriber\_df based on SUB\_ID and sub\_id to include city information. Then it Groups by city, counts claims, orders by count in descending order, and retrieves the city with the most claims.

```

# 7. Join claims_df with subscriber_df to get city information for claims
from pyspark.sql.functions import count, avg, year, current_date, desc
claims_with_city_df = claims_df.join(subscriber_df, claims_df.SUB_ID == subscriber_df["sub_id"], "left")

# City with most claims
city_most_claims = claims_with_city_df.groupBy("City").count().orderBy(desc("count")).first()
print(f"City with most claims: {city_most_claims['City']}")

# Convert to DataFrame
city_most_claims_df = spark.createDataFrame([city_most_claims])

# Display the result
city_most_claims_df.show()

#Saving as a permanent table
permanent_table_name = "city_most_claims"
city_most_claims_df.write.format("csv").mode("overwrite").saveAsTable(permanent_table_name)

# Saving data to AWS S3
city_most_claims_df.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/use-case-output/city_most_claims")

```

▶ (7) Spark Jobs

```

▶ [claims_with_city_df: pyspark.sql.dataframe.DataFrame = [Claim_Or_Rejected: string, SUB_ID: string ... 20 more fields]
▶ [city_most_claims_df: pyspark.sql.dataframe.DataFrame = [City: string, count: long]

City with most claims: Mysore
+-----+
| City|count|
+-----+
|Mysore|     2|
+-----+

```

- **Policy Type Analysis:** Identify whether subscribers mostly subscribe to government or private policies.

This script Groups subscriber\_df by Elig\_ind (government or private policy indicator) and counts the number of subscriptions for each and displays the table of government and private policy subscriptions sorted by count.

```

# 8. Government or private policy subscriptions
policy_type_subscription = subscriber_df.groupBy("Elig_ind").count().orderBy(desc("count"))
policy_type_subscription.show()

#Saving as a permanent table
permanent_table_name = "policy_type_subscription"
policy_type_subscription.write.format("csv").mode("overwrite").saveAsTable(permanent_table_name)

# Saving data to AWS S3
policy_type_subscription.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/use-case-output/policy_type_subscription")

```

▶ (6) Spark Jobs

```

▶ [policy_type_subscription: pyspark.sql.dataframe.DataFrame = [Elig_ind: string, count: long]

+-----+
|Elig_ind|count|
+-----+
| N|    50|
| Y|    46|
| null|     4|
+-----+

```

- **Premium Analysis:** Calculate the average monthly premium paid by subscribers to the insurance company.

This script uses subgroup\_df to calculate the average monthly premium paid by subscribers using avg("Monthly\_Premium")

```

# 9. Average monthly premium paid by subscribers
avg_monthly_premium = subgroup_df.select(avg("Monthly_Premium")).first()
print(f"Average monthly premium paid by subscribers: {avg_monthly_premium['avg(Monthly_Premium)']}")

# Convert to DataFrame
avg_monthly_premium_df = spark.createDataFrame([avg_monthly_premium])

# Display the result
avg_monthly_premium_df.show()

#Saving as a permanent table
permanent_table_name = "avg_monthly_premium"
avg_monthly_premium_df.write.format("csv").mode("overwrite").saveAsTable(permanent_table_name)

# Saving data to AWS S3
avg_monthly_premium_df.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/use-case-output/
avg_monthly_premium")

▶ (6) Spark Jobs
▶ avg_monthly_premium_df: pyspark.sql.dataframe.DataFrame = [avg(Monthly_Premium): double]
Average monthly premium paid by subscribers: 1870.0
+-----+
|avg(Monthly_Premium)|
+-----+
|          1870.0|
+-----+

```

- **Profitability Analysis:** Find out which group is most profitable.

This script calculates the total premium written for each group in group\_df, identifies the group with the highest total premium, and prints its Grp\_Id. It uses aggregation functions like sum() and orderBy() to determine and display the most profitable group based on premium contributions.

```

# 10. Most profitable group

most_profitable_group = group_df.groupBy("Grp_Id") \
    .agg(sum("premium_written").alias("total_premium")) \
    .orderBy(col("total_premium").desc()) \
    .first()

print("Most profitable group:", most_profitable_group["Grp_Id"])

# Convert to DataFrame
most_profitable_group_df = spark.createDataFrame([most_profitable_group])

# Display the result
most_profitable_group_df.show()

#Saving as a permanent table
permanent_table_name = "most_profitable_group"
most_profitable_group_df.write.format("csv").mode("overwrite").saveAsTable(permanent_table_name)

# Saving data to AWS S3
most_profitable_group_df.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/use-case-output/
most_profitable_group")

▶ (6) Spark Jobs
▶ most_profitable_group_df: pyspark.sql.dataframe.DataFrame = [Grp_Id: string, total_premium: long]
Most profitable group: GRP131
+-----+
|Grp_Id|total_premium|
+-----+
|GRP131|      99000|
+-----+

```

- **Pediatric Cancer Patients:** List all patients below age 18 who are admitted for cancer.

This script uses filter() on patient\_records\_df to select records where disease\_name contains "cancer" and the patient's age is below 18 and it shows patients below 18 admitted for cancer.

```
# 11. Patients below age of 18 admitted for cancer
from pyspark.sql.functions import sum as spark_sum

# Filter patients below 18 with cancer
patients_below_18_cancer = patient_records_df.filter(
    (col("disease_name").contains("cancer")) &
    (datediff(current_date(), col("patient_birth_date")) / 365 < 18)
)

# Display the result
patients_below_18_cancer.show()

#Saving as a permanent table
permanent_table_name = "patients_below_18_cancer"
patients_below_18_cancer.write.format("csv").mode("overwrite").saveAsTable(permanent_table_name)

# Saving data to AWS S3
patients_below_18_cancer.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/use-case-output/
patients_below_18_cancer")

▶ (2) Spark Jobs
▶ patients_below_18_cancer: pyspark.sql.dataframe.DataFrame = [Patient_id: integer, Patient_name: string ... 6 more fields]
+-----+-----+-----+-----+-----+-----+
|Patient_id|Patient_name|patient_gender|patient_birth_date|patient_phone|disease_name|city|hospital_id|
+-----+-----+-----+-----+-----+-----+
```

- **High-Value Cashless Insurance Patients:** List patients who have cashless insurance and total charges greater than or equal to Rs. 50,000.

This script performs a join between patient\_records\_df\_cleaned and claims\_df\_cleaned based on Patient\_id then filters the joined DataFrame to include only records where claim\_type is "claims of value" and claim\_amount is 50,000 or more. Finally, selects the necessary columns from both DataFrames (patient\_records\_df\_cleaned and claims\_df\_cleaned) to include in the patients\_cashless DataFrame.

```

#12 List patients who have cashless insurance and have total charges greater than or equal for Rs. 50,000.
# Joining patients and claims DataFrames
joined_df = patient_records_df_cleaned.join(claims_df_cleaned, patient_records_df_cleaned["Patient_id"] == claims_df_cleaned["patient_id"])
# Applying filters
filtered_df = joined_df.filter(
    (claims_df_cleaned["claim_type"] == "claims of value") &
    (claims_df_cleaned["claim_amount"].cast("int") >= 50000)
)
# Selecting required columns
patients_cashless = filtered_df.select(
    patient_records_df_cleaned["Patient_id"],
    patient_records_df_cleaned["Patient_name"],
    patient_records_df_cleaned["hospital_id"],
    claims_df_cleaned["claim_amount"],
    claims_df_cleaned["claim_type"]
)
# Showing the result
patients_cashless.show()
# Saving as a permanent table
permanent_table_name = "patients_cashless"
patients_cashless.write.format("csv").mode("overwrite").saveAsTable(permanent_table_name)

# Saving data to AWS S3
patients_cashless.write.option("header", "true").mode("overwrite").csv("s3a://health-insurance-capstone/use-case-output/patients_cashless")

```

(8) Spark Jobs

- ▶ joined\_df: pyspark.sql.dataframe.DataFrame = [Patient\_id: integer, Patient\_name: string ... 14 more fields]
- ▶ filtered\_df: pyspark.sql.dataframe.DataFrame = [Patient\_id: integer, Patient\_name: string ... 14 more fields]
- ▶ patients\_cashless: pyspark.sql.dataframe.DataFrame = [Patient\_id: integer, Patient\_name: string ... 3 more fields]

Patient_id	Patient_name	Hospital_id	Claim_Amount	Claim_Type
189996	Ekant	H1003	192381	claims of value
109251	Anjushree	H1001	116937	claims of value
167340	NA	H1003	118628	claims of value
132947	Saroj	H1016	186502	claims of value
172579	Devnath	H1019	168634	claims of value
109342	Chitraranjan	H1011	125727	claims of value
1463821	Dharmadaas	H1019	75983	claims of value

- **Female Knee Surgery Patients:** List female patients over the age of 40 that have undergone knee surgery in the past year.

The script filters female patients over 40 years old from cleaned patient records (patient\_records\_df\_cleaned). It further refines this group to include only those who underwent knee surgery, displaying the results and saving them as a permanent CSV table named "knee\_surgery\_patients" in Spark.

Just now (2s) 30

```
# 13. Female patients over age 40 who underwent knee surgery in the past year
from pyspark.sql.functions import col, year, datediff, current_date

# Filter for female patients over the age of 40
female_patients_over_40 = patient_records_df.filter(
    (col("patient_gender") == "Female") &
    (datediff(current_date(), col("patient_birth_date")) / 365 > 40)
)
#female_patients_over_40.show()
# Filter for knee surgery in the past year
knee_surgery_patients = female_patients_over_40.filter(
    col("disease_name").contains("Knee Surgery"))

# Show the results
knee_surgery_patients.show()

# #Saving as a permanent table
permanent_table_name = "knee_surgery_patients"
knee_surgery_patients.write.format("csv").mode("overwrite").saveAsTable(permanent_table_name)

▶ (2) Spark Jobs
▶ female_patients_over_40: pyspark.sql.dataframe.DataFrame = [Patient_id: integer, Patient_name: string ... 6 more fields]
▶ knee_surgery_patients: pyspark.sql.dataframe.DataFrame = [Patient_id: integer, Patient_name: string ... 6 more fields]
+-----+-----+-----+-----+-----+-----+
|Patient_id|Patient_name|patient_gender|patient_birth_date|patient_phone|disease_name|city|hospital_id|
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
```

## 12. Result Creation on Redshift

The screenshot shows the AWS S3 console interface. On the left, there's a navigation sidebar with links like 'Amazon S3', 'Buckets', 'Access Grants', 'Access Points', etc. The main area shows a list of objects in a folder named 'use-case-output/'. The list includes 12 items, all of which are folders. The columns in the table are 'Name', 'Type', 'Last modified', 'Size', and 'Storage class'. The 'Actions' menu is visible at the top of the list.

Name	Type	Last modified	Size	Storage class
avg_monthly_premium/_	Folder	-	-	-
city_most_claims/_	Folder	-	-	-
knee_surgery_patients/_	Folder	-	-	-
max_claims_disease/_	Folder	-	-	-
max_subgroup_group/_	Folder	-	-	-
most_profitable_group/_	Folder	-	-	-
most_subscribed_subgroup/_	Folder	-	-	-
patients_below_18_cancer/_	Folder	-	-	-
patients_cashless/_	Folder	-	-	-
policy_type_subscription/_	Folder	-	-	-
subscribers_under_30/_	Folder	-	-	-
total_rejected_claims/_	Folder	-	-	-

**Redshift query editor v2**

Editor    Create    Load data    Run    Limit 100    Explain    Isolated session    Serverless: he...    dev    Schedule    ...

Filter resources: Serverless: healthcarewg  
 > awssdatacatalog  
 > dev  
 > project-output  
 > Tables  
 avg\_monthly\_premium  
 city\_most\_claims  
 knee\_surgery\_patients  
 max\_claims\_disease  
 max\_subgroup\_group  
 most\_profitable\_group  
 most\_subscribed\_subgroup  
 patients\_below\_18\_cancer  
 patients\_cashless  
 policy\_type\_subscription  
**subscribers\_under\_30**  
 total\_rejected\_claims  
 > Views  
 0

**subscribers\_under\_30**

Field	Type	NL	CMP
A sub_id	character varying(256)	NULL	Izo
A first_name	character varying(256)	NULL	Izo
A last_name	character varying(256)	NULL	Izo
A street	character varying(256)	NULL	Izo
B birth_date	date	NULL	az64
A gender	character varying(256)	NULL	Izo

Result 1 (3)

sub_id	first_name	last_name	street	birth_date	gender	phone
SUBID10017	Bandhu	Seth	Varughese	1996-10-15	Male	+91 08952891
SUBID10083	Bhilangana	Pandit	Ramachandran Path	1995-01-04	Female	+91 65530666
SUBID10093	Chandavarman	Singh	Sarkar Circle	1997-05-10	Others	+91 65590317

Row 1, Col 52, Chr 52    Export    Chart    Row 1, Col 52, Chr 52

Query ID 3925    Elapsed time: 15 ms    Total rows: 3

**databricks**

**New**

- Workspace
- Recents
- Search
- Catalog**
- Workflows
- Compute

Machine Learning

Experiments

**Data**

Create Table    X   

Databases   

Filter Databases

default

Tables

Filter Tables

- avg\_montshly\_premium
- city\_mosts\_claims
- knee\_surgresy\_patients
- max\_claims\_disease
- max\_claims\_disease
- max\_subgraoup\_group
- mosst\_profitable\_group
- mosst\_subscribed\_subgroup
- patiesnts\_below\_18\_cancer
- patiesnts\_cashless
- policy\_type\_susbscription
- subsscribers\_under\_30\_df
- total\_rejeected\_claims