  
  
A screenshot of a computer

Description automatically generated  
  
A screenshot of a computer

Description automatically generated  
  
A screenshot of a computer

Description automatically generated  
  
A screenshot of a computer

Description automatically generated  
  
A screenshot of a computer

Description automatically generated  
  
A screenshot of a computer

Description automatically generated  
  
Null columns counts  
A screenshot of a computer

Description automatically generated  
  
  
A screenshot of a computer

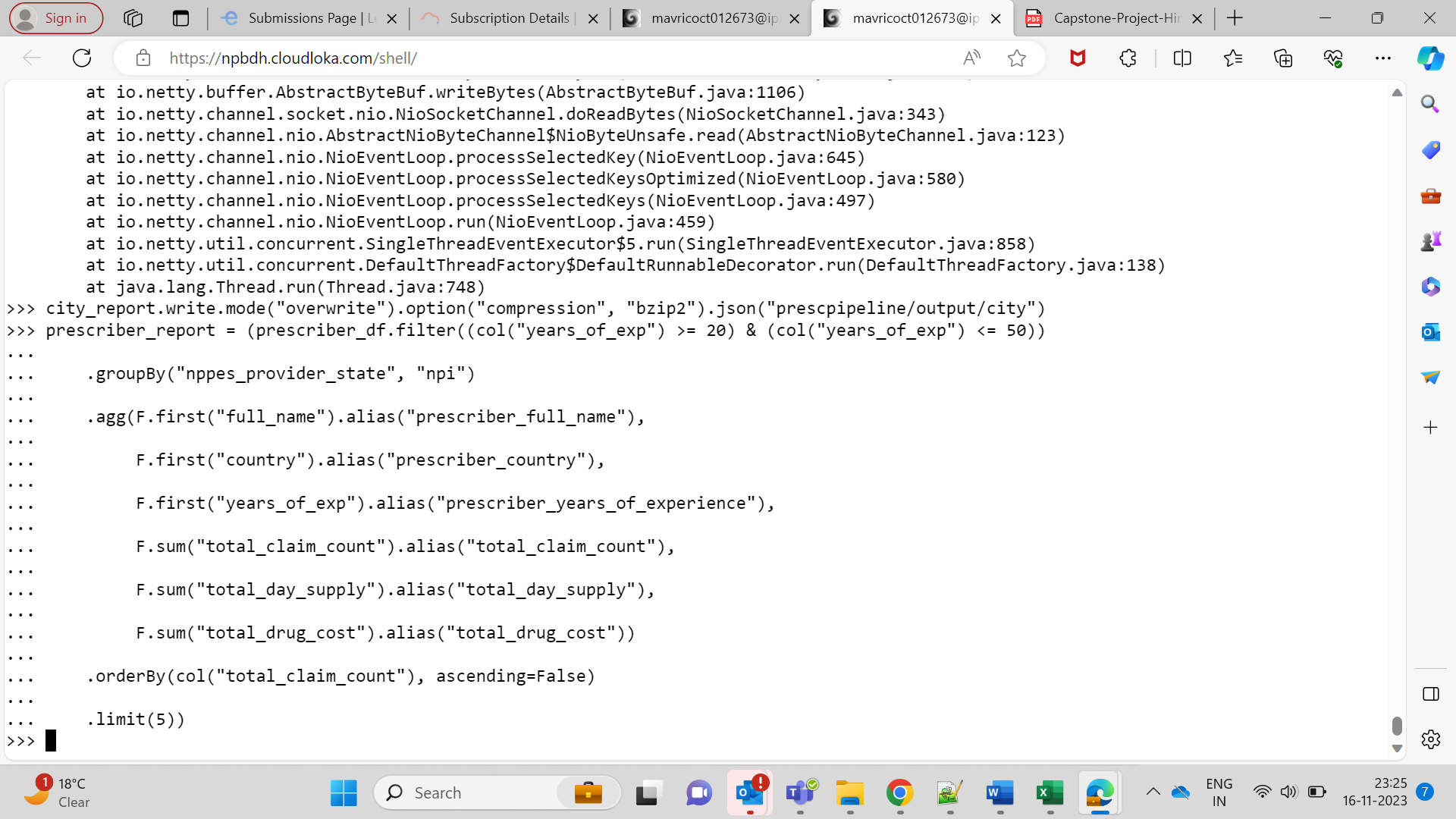
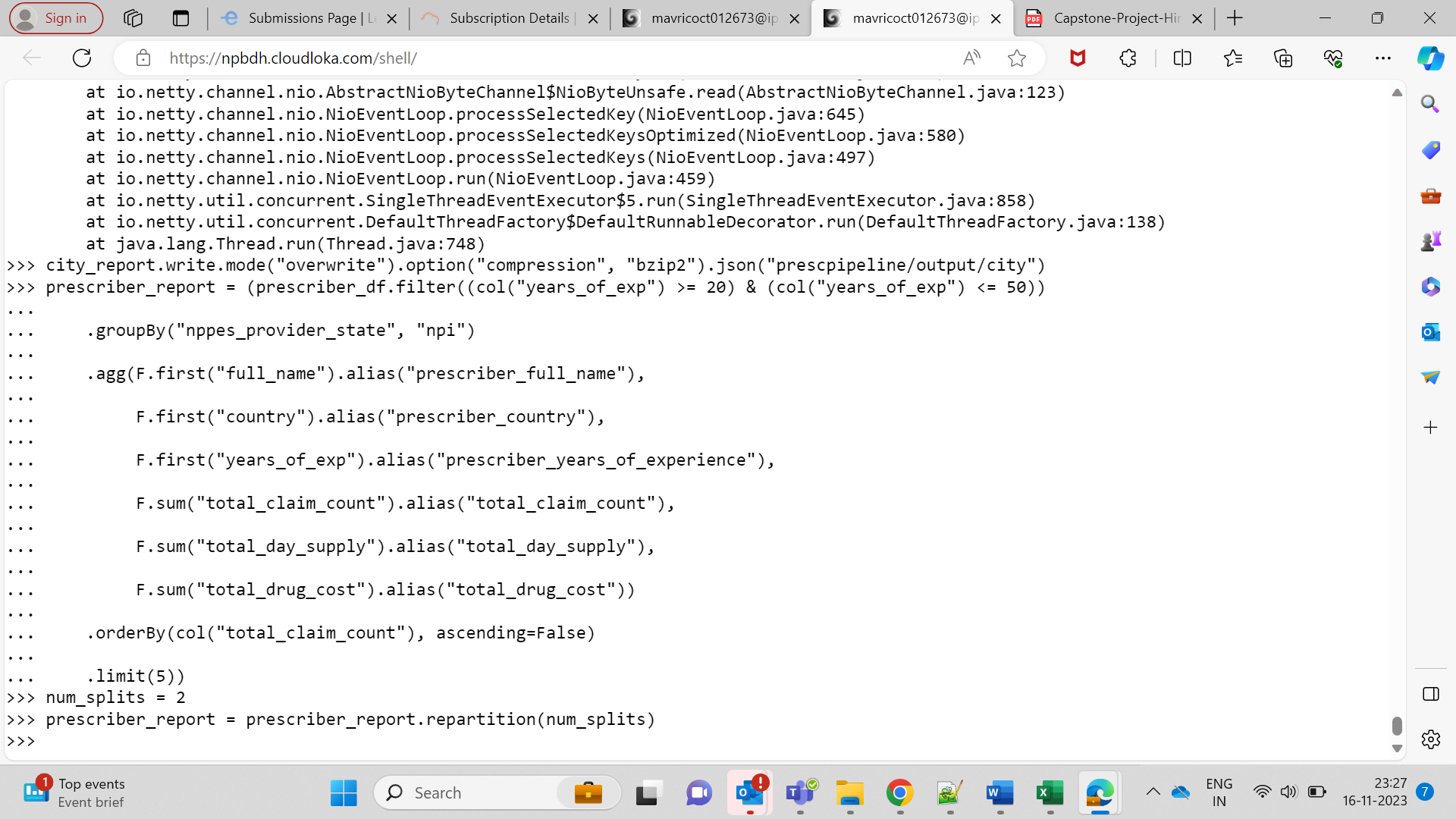
Description automatically generated  
  
  
A screenshot of a computer

Description automatically generated  
  
A screenshot of a computer

Description automatically generated  
  
A screenshot of a computer

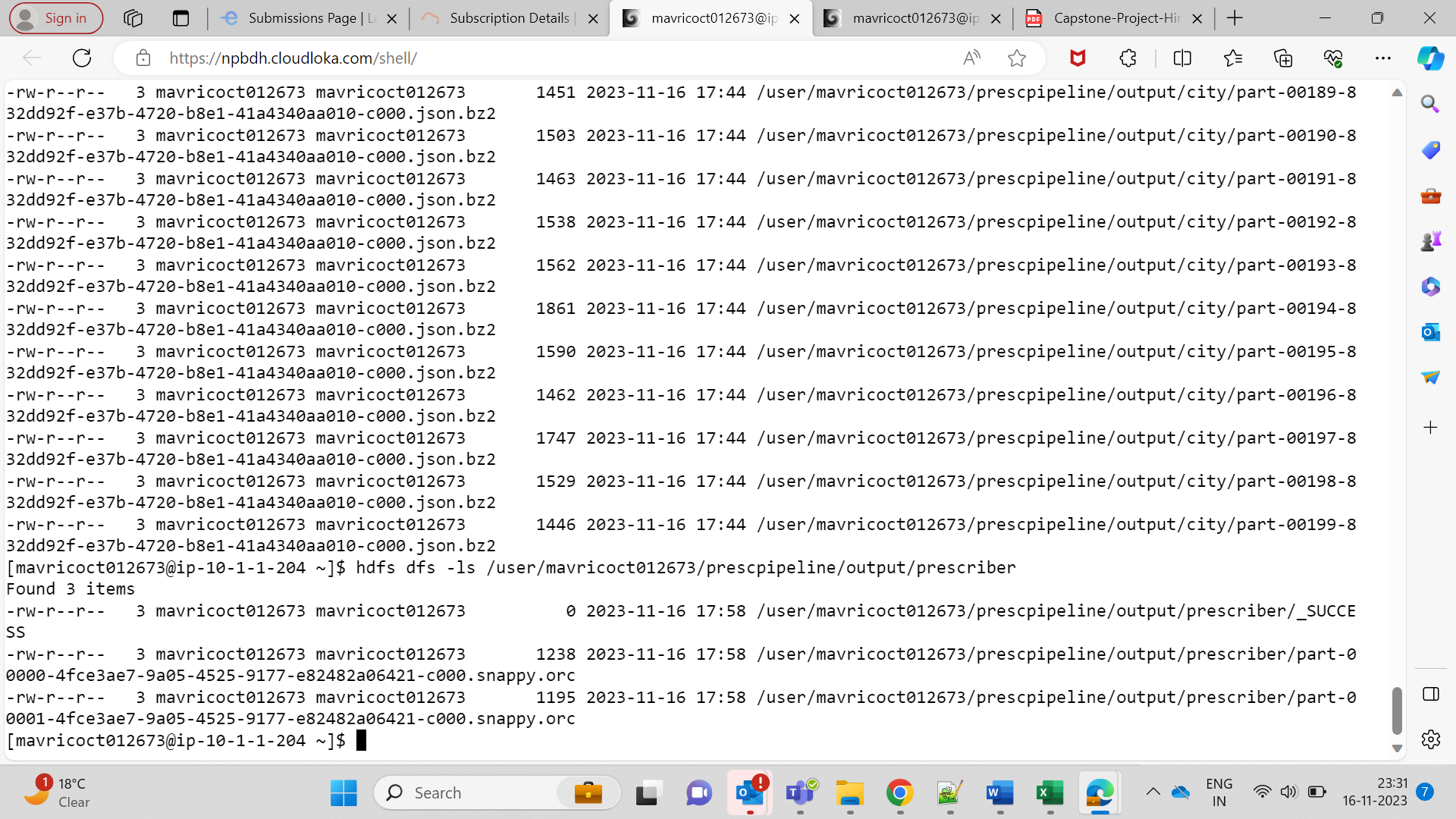
Description automatically generated  
  
  
A screenshot of a computer

Description automatically generated  
  
  
A screenshot of a computer

Description automatically generated  
  
**Prescriber Report-**  
  
  
  
  
  
A screenshot of a computer

Description automatically generated  
  
A screenshot of a computer

Description automatically generated  
  
A screenshot of a computer

Description automatically generated  
  
  
  
A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated  
  
  
from pyspark.sql import SparkSession

from pyspark.sql.functions import col, countDistinct, sum, size, split

from pyspark.sql.types import IntegerType

import pyspark.sql.functions as F

# Step 1: Initialize Spark session

spark = SparkSession.builder.appName("HealthcareETL").getOrCreate()

# Step 2: Data Ingestion

city\_df = spark.read.option("inferSchema", "true").parquet("prescpipeline/staging/city/us\_cities\_dimension.parquet")

prescriber\_df = spark.read.csv("prescpipeline/staging/prescriber/USA\_Presc\_Medicare\_Data\_12021.csv", header=True, inferSchema=True)

# Step 3: Data Cleansing

# Clean City Data

city\_df = (city\_df

.select("city", "state\_id", "state\_name", "county\_name", "population", "zips")

.withColumn("city", col("city").cast("string"))

.withColumn("state\_id", col("state\_id").cast("string"))

.withColumn("state\_name", col("state\_name").cast("string"))

.withColumn("county\_name", col("county\_name").cast("string"))

.withColumn("population", col("population").cast("integer"))

.withColumn("zips", col("zips").cast("string"))

.withColumn("number\_of\_zips", size(split(col("zips"), ","))

)

# Clean Prescriber Data

prescriber\_df = prescriber\_df.select("npi", "nppes\_provider\_last\_org\_name", "nppes\_provider\_first\_name",

"nppes\_provider\_city", "nppes\_provider\_state", "specialty\_description",

"drug\_name", "total\_claim\_count", "total\_day\_supply", "total\_drug\_cost") \

.withColumnRenamed("nppes\_provider\_last\_org\_name", "last\_name") \

.withColumnRenamed("nppes\_provider\_first\_name", "first\_name") \

.withColumn("country", F.lit("USA")) \

.withColumn("years\_of\_exp", F.regexp\_extract(col("years\_of\_exp"), r"(\d+)", 1).cast(IntegerType())) \

.withColumn("full\_name", F.concat(col("first\_name"), F.lit(" "), col("last\_name"))) \

.na.drop(subset=["npi", "drug\_name"])

# Count the number of null values for each column

null\_counts = prescriber\_df.select([count(when(col(c).isNull() | isnan(col(c)), c)).alias(c) for c in prescriber\_df.columns])

# Clean all the Null/Nan Values

prescriber\_df = prescriber\_df.na.drop(subset=["npi", "drug\_name"])

# Step 4: Data Transformation for City Report

# Calculate the number of zips in each city

city\_df = city\_df.withColumn("number\_of\_zips", size(split(col("zips"), ",")))

# Calculate the number of distinct Prescribers assigned for each City

prescriber\_counts = prescriber\_df.groupBy("nppes\_provider\_state", "nppes\_provider\_city").agg(

countDistinct("npi").alias("prescriber\_counts"),

sum(col("total\_claim\_count").cast("int")).alias("total\_claim\_counts")

)

# Join dataframes using state\_id and nppes\_provider\_state

city\_report = city\_df.join(

prescriber\_counts,

(city\_df.state\_id == prescriber\_counts.nppes\_provider\_state) & (city\_df.city == prescriber\_counts.nppes\_provider\_city),

"left\_outer"

)

city\_report = city\_df.join(

prescriber\_counts,

(upper(city\_df.state\_id) == upper(prescriber\_counts.nppes\_provider\_state)) &

(upper(city\_df.city) == upper(prescriber\_counts.nppes\_provider\_city)),

"left\_outer"

)

# Filter out cities with no prescriber assigned

city\_report = city\_report.filter(city\_report.prescriber\_counts > 0)

# Select the required columns

city\_report = city\_report.select("city", "state\_id", "state\_name", "county\_name", "population", "number\_of\_zips", "prescriber\_counts", "total\_claim\_counts")

# Show the final City Report

city\_report.show()

# Save the City Report as JSON with Bzip2 compression

city\_report.write.mode("overwrite").option("compression", "bzip2").json("prescpipeline/output/city")

# Step 4: Data Transformation for Prescriber Report

prescriber\_report = (prescriber\_df.filter((col("years\_of\_exp") >= 20) & (col("years\_of\_exp") <= 50))

.groupBy("nppes\_provider\_state", "npi")

.agg(F.first("full\_name").alias("prescriber\_full\_name"),

F.first("country").alias("prescriber\_country"),

F.first("years\_of\_exp").alias("prescriber\_years\_of\_experience"),

F.sum("total\_claim\_count").alias("total\_claim\_count"),

F.sum("total\_day\_supply").alias("total\_day\_supply"),

F.sum("total\_drug\_cost").alias("total\_drug\_cost"))

.orderBy(col("total\_claim\_count"), ascending=False)

.limit(5))

# Set the desired number of splits (output files)

num\_splits = 2

# Repartition the DataFrame to control the number of output splits

prescriber\_report = prescriber\_report.repartition(num\_splits)

# Write the Prescriber Report in ORC format with Snappy compression and overwrite the output directory if it exists

prescriber\_report.write.mode("overwrite").orc("prescpipeline/output/prescriber")

from pyspark.sql import SparkSession

# Initialize a Spark session

spark = SparkSession.builder.appName("OpenORCFile").getOrCreate()

# Specify the path to the ORC file

orc\_file\_path = "prescpipeline/output/prescriber"

# Read the ORC file

prescriber\_report = spark.read.format("orc").load(orc\_file\_path)

# Show the content of the ORC file

prescriber\_report.show()

# Step 6: Stop the Spark session

spark.stop()