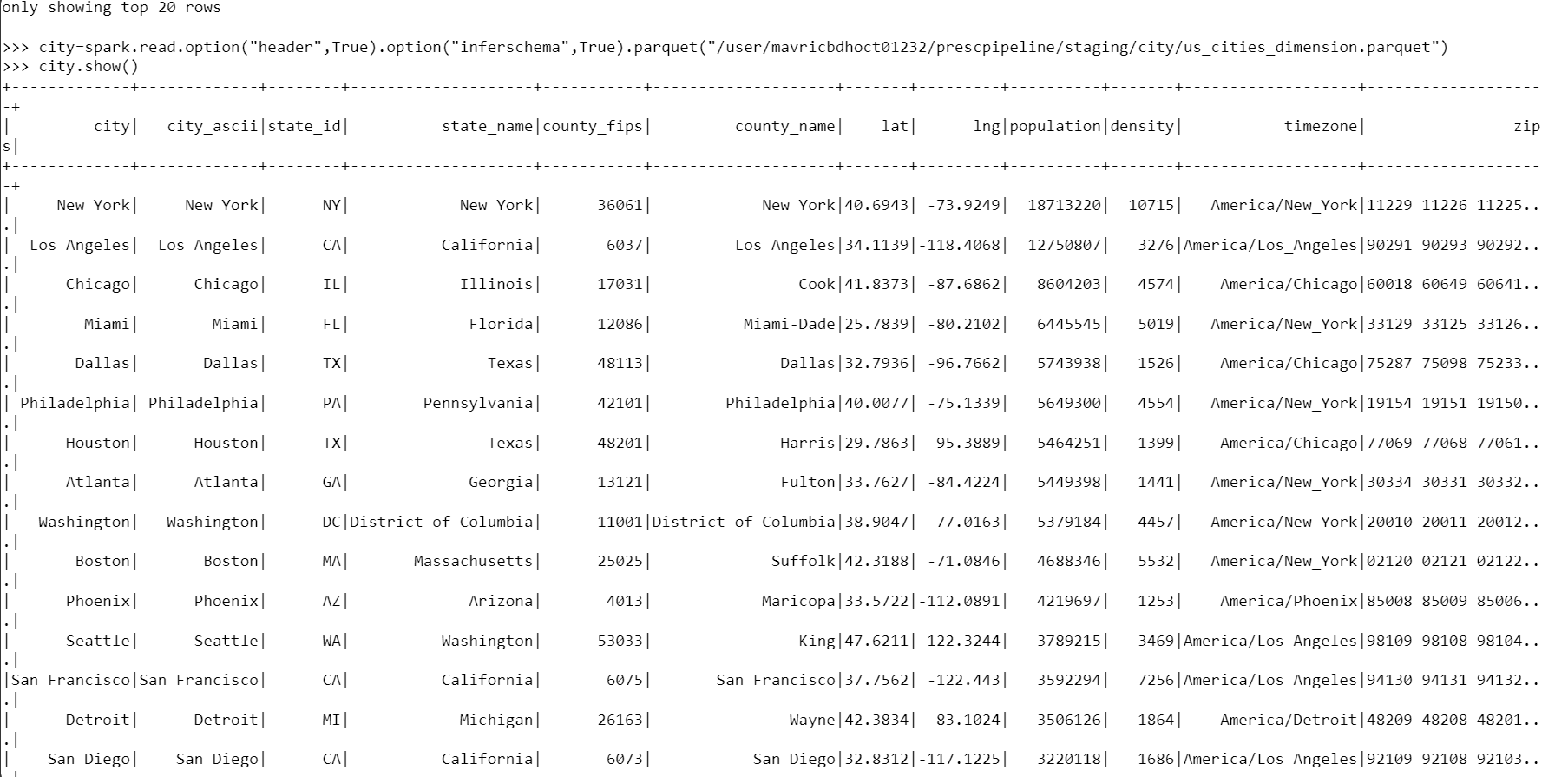
Input City Data format/Layout



Input Prescriber Data Layout

A close up of a document

Description automatically generated

Output data layout (City)

df = spark.read.table("city")

num\_splits = 1

output\_format = "JSON"

compression = "Bzip2"

spark.conf.set("mapreduce.input.fileinputformat.split.maxsize", f"{df.rdd.getNumPartitions() \* num\_splits}")

spark.sql("SET hive.exec.orc.default.compress=SNAPPY")

df.write.format(output\_format).mode("overwrite").option("compression",compression).saveAsTable("Gradesh\_Db\_Airline.city")

Output data layout (Prescriber)

df = spark.read.table("pres1")

num\_splits = 2

output\_format = "orc"

compression = "snappy"

spark.conf.set("mapreduce.input.fileinputformat.split.maxsize", f"{df.rdd.getNumPartitions() \* num\_splits}")

spark.sql("SET hive.exec.orc.default.compress=SNAPPY")

df.write.format(output\_format).mode("overwrite").option("compression", compression).saveAsTable("Gradesh\_Db\_Airline.pres")

For City

df = spark.read.table("data")

num\_splits = 1

output\_format = "JSON"

compression = "Bzip2"

spark.conf.set("mapreduce.input.fileinputformat.split.maxsize", f"{df.rdd.getNumPartitions() \* num\_splits}")

spark.sql("SET hive.exec.orc.default.compress=SNAPPY")

df.write.format(output\_format).mode("overwrite").option("compression", compression).saveAsTable("Hivecity ")

For Prescriber

df = spark.read.table("pres1")

num\_splits = 2

output\_format = "orc"

compression = "snappy"

spark.conf.set("mapreduce.input.fileinputformat.split.maxsize", f"{df.rdd.getNumPartitions() \* num\_splits}")

spark.sql("SET hive.exec.orc.default.compress=SNAPPY")

df.write.format(output\_format).mode("overwrite").option("compression", compression).saveAsTable("airline\_db.pres")

for Cleansing (City)

**cityDF1=cityDF.select('city','state\_id','state\_name','county\_name','population','zips')**

**cityDF1.show(4)**

**cityDF1.printSchema()**

A close-up of a number

Description automatically generated

For Prescriber ()

**Converting the required fields to upper case** For City

import pyspark.sql.functions as F  
city=spark.read.option("header",True).option("inferschema",True).parquet('prescpipeline/staging/city/us\_cities\_dimension.parquet')  
cityDF = city.select('city','state\_id','state\_name','county\_name','population','zips')  
cityDF = city.select(F.upper(city.city).alias("city"),'state\_id',F.upper(city.state\_name).alias("state\_name"),F.upper(city.county\_name).alias("county\_name"),'population','zips')  
from pyspark.sql.functions import size  
from pyspark.sql.functions import split  
df2 = cityDF.withColumn('zips',size(split(cityDF['zips']," ")))   
cityDF.createOrReplaceTempView("city\_data")

**For Prescriber**

prescriberDF=spark.read.option("header",True).option("inferschema",True).csv("prescpipeline/staging/prescriber/USA\_Presc\_Medicare\_Data\_12021.csv")   
prescriberDF.show(10)  
prescriberDF.printSchema()  
df1=prescriberDF.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')     
df1=prescriberDF.select('npi',(prescriberDF.nppes\_provider\_last\_org\_name).alias ('np\_Lname'),(prescriberDF.nppes\_provider\_first\_name).alias ('np\_Fname'),(prescriberDF.nppes\_provider\_city).alias ('np\_city'),(prescriberDF.nppes\_provider\_state).alias ('np\_state'),(prescriberDF.specialty\_description).alias ('speciality\_desc'),(prescriberDF.drug\_name).alias ('drug\_nm'),(prescriberDF.total\_claim\_count).alias ('tot\_clm\_count'),(prescriberDF.total\_day\_supply).alias ('tot\_day\_supp'),(prescriberDF.total\_drug\_cost) .alias ('tot\_drug\_cst'))      
from pyspark.sql.functions import col, lit  
df1=df1.withColumn("Country", lit("USA"))   
PrescriberDF1.show(3)  
df1=prescriberDF.select('npi',(prescriberDF.nppes\_provider\_last\_org\_name).alias ('np\_Lname'),(prescriberDF.nppes\_provider\_first\_name).alias ('np\_Fname'),  
(prescriberDF.nppes\_provider\_city).alias ('np\_city'),(prescriberDF.nppes\_provider\_state).alias ('np\_state'),(prescriberDF.specialty\_description).alias ('speciality\_desc'),  
(prescriberDF.drug\_name).alias ('drug\_nm'),(prescriberDF.total\_claim\_count).alias ('tot\_clm\_count'),(prescriberDF.total\_day\_supply).alias ('tot\_day\_supp'),  
(prescriberDF.total\_drug\_cost) .alias ('tot\_drug\_cst'),'years\_of\_exp')     
import pyspark.sql.functions as F  
df1= df1.withColumn("years\_of\_exp", F.regexp\_replace("years\_of\_exp", "=", ""))                                       
df1= df1.withColumn("years\_of\_exp", F.regexp\_replace("years\_of\_exp", "\.0", ""))  
df1= df1.withColumn("years\_of\_exp", F.regexp\_replace("years\_of\_exp", " ", ""))                                                                                                    
from pyspark.sql.functions import concat, concat\_ws,col, lit    
df1 = df1.withColumn("years\_of\_exp",col("years\_of\_exp").cast("Integer"))       
from pyspark.sql.functions import concat, concat\_ws,col, lit                                                                             
df1 = df1.withColumn("presc\_fullname",concat\_ws(" ", "np\_Fname", "np\_Lname"))                                        
df1 = df1.drop("np\_Fname", "np\_Lname")                                                                                  
from pyspark.sql.functions import col,isnan, when, count    
df1 = df1.dropna(subset = 'npi')                                                                                                     
df1 = df1.dropna(subset = 'drug\_nm')  
df1 = df1.dropna(subset = 'tot\_clm\_count')                                                                                                 
df1.select([count(F.when(isnan(c) | col(c).isNull(),c)).alias(c) for c in df1.columns]).show()  
df1.createOrReplaceTempView("pres")

**Transform Logic: City Report**

**from pyspark.sql.functions import col, lit**

**from pyspark.sql.functions import split, explode , size**

**split\_col=split(cityDF1['ZIPS'],' ')**

**df\_split = cityDF1.withColumn('Zip\_Split\_cnt',size(split\_col))**

**df\_split.createOrReplaceTempView("City\_split\_df")**

**A screenshot of a computer

Description automatically generated**

**Calculate the number of distinct Prescribers assigned for each City.**

**prescriber1\_df = spark.sql("SELECT count(distinct npi) as Prescriber\_Counts,np\_city from prescriber group by np\_city").show()**

**A screenshot of a computer

Description automatically generated**

**Calculate total total\_claim\_count prescribed for each city.**

**prescriber1\_df = spark.sql("SELECT sum(tot\_claim\_cnt) as total\_claim\_counts, np\_state from prescriber group by np\_state") .show()**

**A close-up of a number

Description automatically generated**

**Do not report a city in the final report if no prescriber is assigned to it.**

prescriber1\_df.createOrReplaceTempView("prescriber1")

df\_split.createOrReplaceTempView("City\_split\_df")

spark.sql( "SELECT CITY as City\_Name,STATE\_NAME as State\_Name,COUNTY\_NAME as County\_Name ,POPULATION as City\_Population,Zip\_Split\_cnt as Number\_of\_Zips,p.Prescriber\_Counts,p.total\_claim\_counts FROM City\_split\_df c inner join prescriber1 p on upper(c.CITY) = p.np\_city and c.STATE\_ID = p.np\_state").show()

spark.sql( "SELECT CITY as City\_Name,STATE\_NAME as State\_Name,COUNTY\_NAME as County\_Name ,POPULATION as City\_Population,Zip\_Split\_cnt as Number\_of\_Zips,p.Prescriber\_Counts,p.total\_claim\_counts FROM City\_split\_df c inner join prescriber1 p on upper(c.CITY) = p.np\_city and c.STATE\_ID = p.np\_state where p.Prescriber\_Counts != 0").show(),

**Checking if any city having no prescriber :**

spark.sql( "SELECT CITY as City\_Name,STATE\_NAME as State\_Name,COUNTY\_NAME as County\_Name ,POPULATION as City\_Population,Zip\_Split\_cnt as Number\_of\_Zips,p.Prescriber\_Counts,p.total\_claim\_counts FROM City\_split\_df c inner join prescriber1 p on upper(c.CITY) = p.np\_city and c.STATE\_ID = p.np\_state where p.Prescriber\_Counts = 0").show()

**A screenshot of a computer

Description automatically generated**

**Writing the City Report outputs in to Hive/HDFS:**

from pyspark.sql import SparkSession

spark=SparkSession.builder.appName("SparkHive").enableHiveSupport().config("spark.sql.warehouse.dir","/user/hive/warehouse").getOrCreate()

spark.sparkContext.setLogLevel("ERROR")

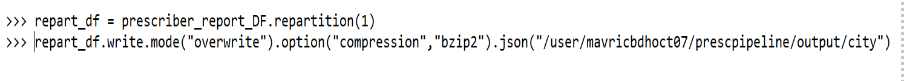
City\_report\_df = spark.sql( "SELECT CITY as City\_Name,STATE\_NAME as State\_Name,COUNTY\_NAME as County\_Name ,POPULATION as

City\_Population,Zip\_Split\_cnt as Number\_of\_Zips,p.Prescriber\_Counts,p.total\_claim\_counts FROM City\_split\_df c inner join prescriber1 p on upper(c.CITY) = p.np\_city and c.STATE\_ID = p.np\_state where p.Prescriber\_Counts != 0")

**No of splits: 1**

repart\_df = prescriber\_report\_DF.repartition(1)

repart\_df.write.mode("overwrite").option("compression","bzip2").json("/user/mavricbdhoct07/prescpipeline/output/city")



**City output In HDFS**

A screenshot of a computer

Description automatically generated

**Select \* from City\_Final\_Report limit 10;**

A screenshot of a computer

Description automatically generated

**select count(\*) from City\_Final\_Report;**

A screenshot of a computer

Description automatically generated