**Project: Health care Data Analysis**

**US City report**

**Data Ingestion**

Creating input Directory:

[maverickavitha@ip-10-1-1-204 ~]$ hdfs dfs -mkdir -p prescpipeline/staging/city

[maverickavitha@ip-10-1-1-204 ~]$ hdfs dfs -put us\_cities\_dimension.parquet prescpipeline/staging/city/

[maverickavitha@ip-10-1-1-204 ~]$ hdfs dfs -ls prescpipeline/staging/city/

A yellow line on a white background

Description automatically generated

**Preprocessing/ Data Cleansing**

Step 1: Creating DF

>>> cityDF=spark.read.option("header",True).option("inferschema",True).parquet ('/user/maverickavitha/prescpipeline/staging/city/us\_cities\_dimension.parquet')

>>> cityDF.printSchema()

>>> cityDF.show(5)

A screenshot of a computer

Description automatically generated

Step 2:

Select only required Columns in city data file like city, state\_id,state\_name,county\_name,population,zips

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

>>> cityDF1.printSchema()

>>> cityDF1.show(3)

A screenshot of a computer

Description automatically generated

Step 3:

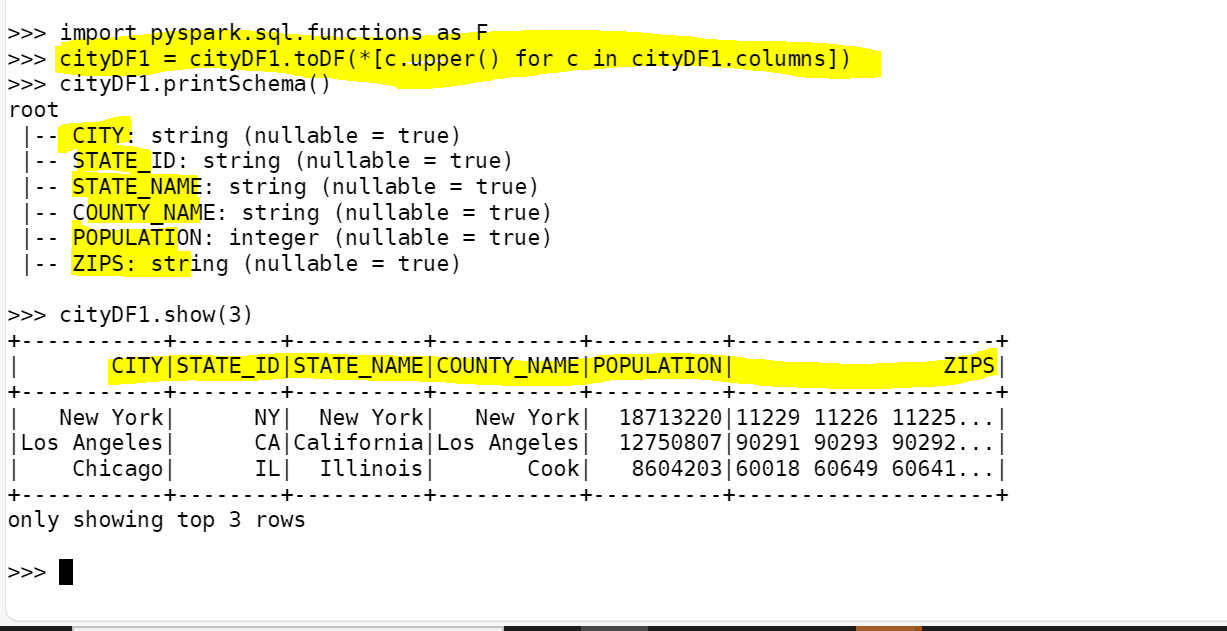
Convert city, state and county fields to Upper Case

>>> import pyspark.sql.functions as F

>>> cityDF1 = cityDF1.toDF(\*[c.upper() for c in cityDF1.columns])

>>> cityDF1.printSchema()

>>> cityDF1.show(3)



**Transformation :**

Calculate the Number of zips in each city.

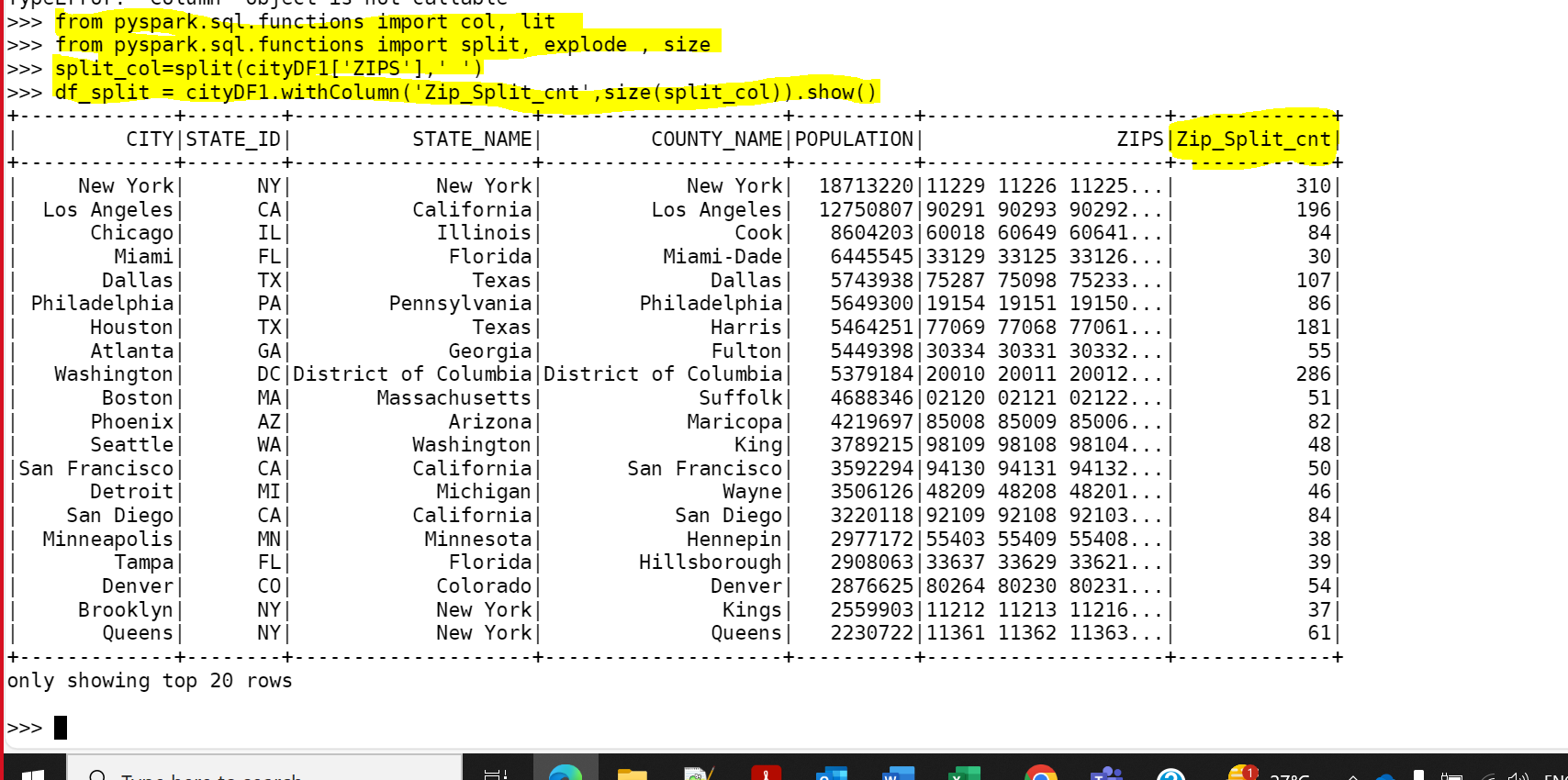
>>> from pyspark.sql.functions import col, lit

>>> 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")



* 1. Calculate the number of distinct Prescribers assigned for each City. Calculate total total\_claim\_count prescribed for each city.

Step 1:

Calculating the distinct prescriber count and total\_claim\_count from Prescribers file , grouping City and state.

>>> spark.sql("SELECT count(distinct npi) as Prescriber\_Counts ,sum(tot\_claim\_cnt) as total\_claim\_counts,np\_city, np\_state from prescr

iber group by np\_city,np\_state").show()

A screenshot of a computer

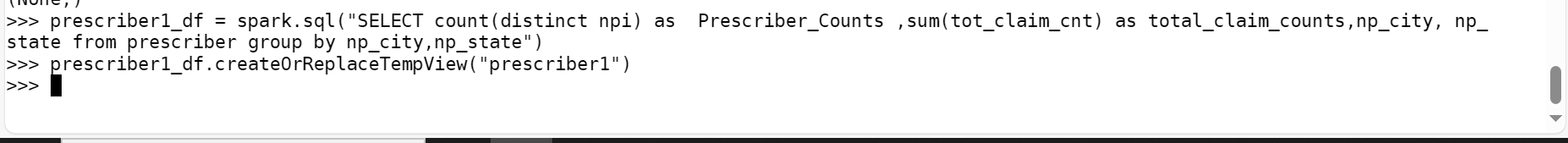
Description automatically generated

Creating Dataframe and view for above prescriber data :

>>> prescriber1\_df = spark.sql("SELECT count(distinct npi) as Prescriber\_Counts ,sum(tot\_claim\_cnt) as total\_claim\_counts,np\_city, np\_

state from prescriber group by np\_city,np\_state")

>>> prescriber1\_df.createOrReplaceTempView("prescriber1")



Joining the City and Prescriber DF to get City Final Report:

>>> 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 where p.Prescriber\_Counts != 0")

A screenshot of a document

Description automatically generated

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

>>> spark.sql( "SELECT CITY as City\_Name,STATE\_NAME as State\_Name,COUNTY\_NAME as County\_Name ,POPULATION as City\_Popul

ation,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")

* 1. A screenshot of a document

     Description automatically generated

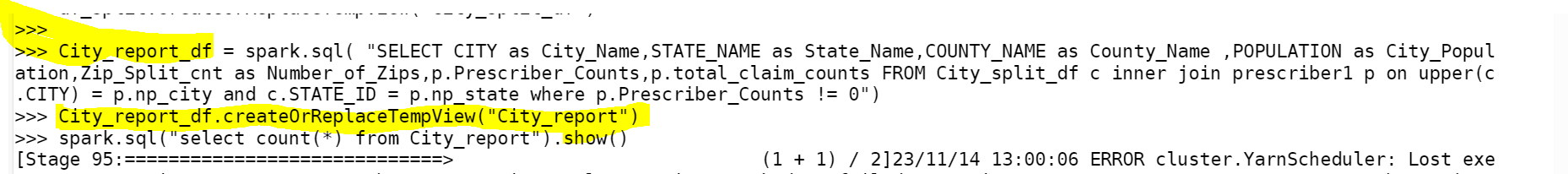
City Final Report Data frame:

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

ation,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")

>>> City\_report\_df.createOrReplaceTempView("City\_report")



Report Layout:

>>> spark.sql("select \* from City\_report").show(5)

A yellow line with numbers and a yellow line

Description automatically generated with medium confidence

Report Data count:

>>> spark.sql("select count(\*) from City\_report").show()

A computer code with text

Description automatically generated

Creating Output Directory in HDFS:

>>> hdfs dfs -mkdir prescpipeline/output/city

A yellow and black text

Description automatically generated

Writing the City Report into HDFS in json format with bzip2 compression

>>> from pyspark.sql import SparkSession

>>> spark=SparkSession.builder.appName("Spark Hive").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")

>>> City\_report\_df.write.mode("overwrite").option("compression","bzip2").json("/user/maverickavitha/prescpipeline/output/city")

A screenshot of a computer

Description automatically generated

City Report Stored in Output file path , in json format with bzip2 compression:

>>> hdfs dfs -ls /user/maverickavitha/prescpipeline/output/city

A yellow line on a white background

Description automatically generated

Writing City\_report into Hive table:

>>> City\_report\_df.write.mode("overwrite").saveAsTable("Final\_City\_Report")

A screenshot of a computer

Description automatically generated

Hive Table Layout

A screenshot of a computer

Description automatically generated

Data in Hive:

select \* from Final\_City\_Report limit 10;

A screenshot of a computer

Description automatically generated

Hive Count:

select count(\*) from Final\_City\_Report;

A screenshot of a computer

Description automatically generated