**Project: Health care Data Analysis**

**Input files in HDFS**

prescpipeline/staging/city

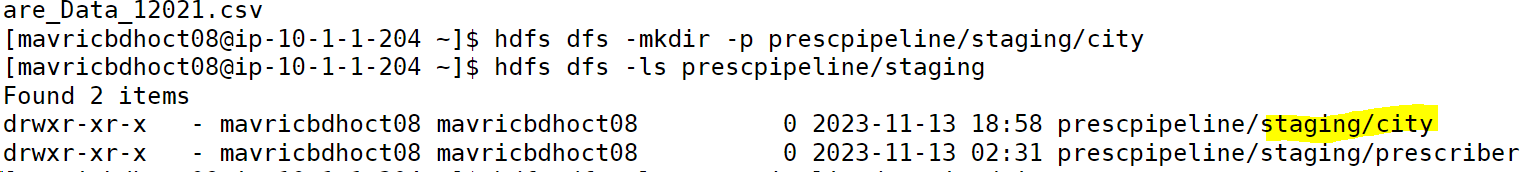
prescpipeline/staging/prescriber

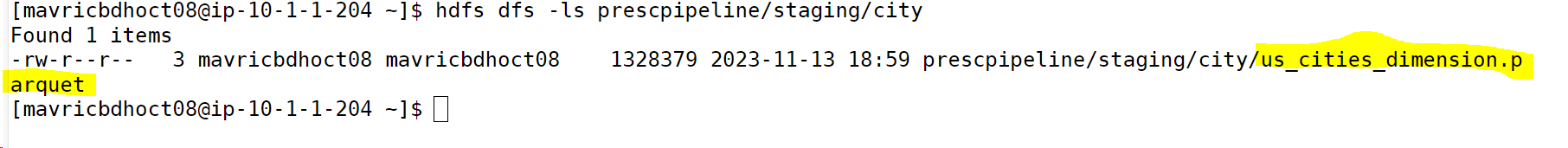
**Data ingestion**

Step1: Creating Input directory and putting files in HDFS.

hdfs dfs -mkdir -p prescpipeline/staging/city

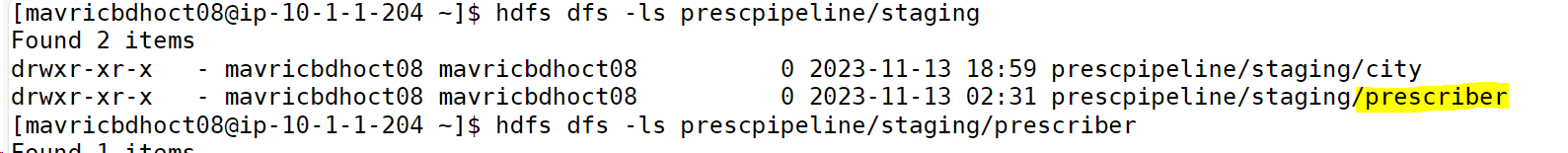
hdfs dfs -put us\_cities\_dimension.parquet prescpipeline/staging/city

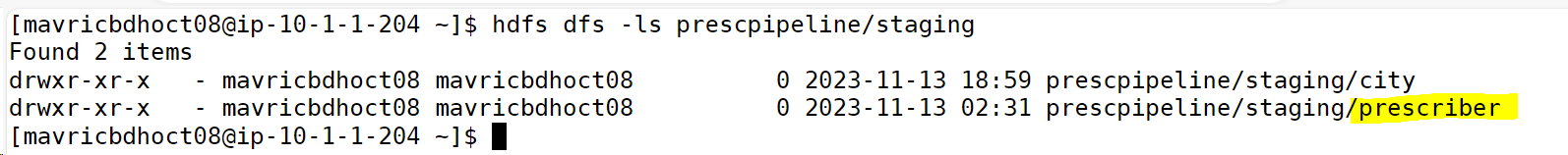


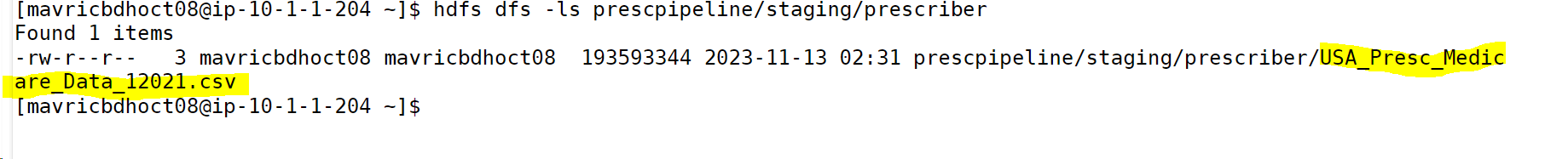


hdfs dfs -mkdir -p prescpipeline/staging/city

hdfs dfs -put us\_cities\_dimension.parquet prescpipeline/staging/city







**Data Preprocessing**

**Clean City Data:**

Step1.

* 1. **Select only required Columns in city data file like city, state\_id,state\_name,county\_name,population,zips**
  2. **Convert city, state and county fields to Upper Case**

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

A screenshot of a computer

Description automatically generated

>>> city\_df.printSchema()

root

|-- city: string (nullable = true)

|-- city\_ascii: string (nullable = true)

|-- state\_id: string (nullable = true)

|-- state\_name: string (nullable = true)

|-- county\_fips: integer (nullable = true)

|-- county\_name: string (nullable = true)

|-- lat: double (nullable = true)

|-- lng: double (nullable = true)

|-- population: integer (nullable = true)

|-- density: integer (nullable = true)

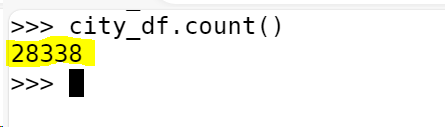
|-- timezone: string (nullable = true)

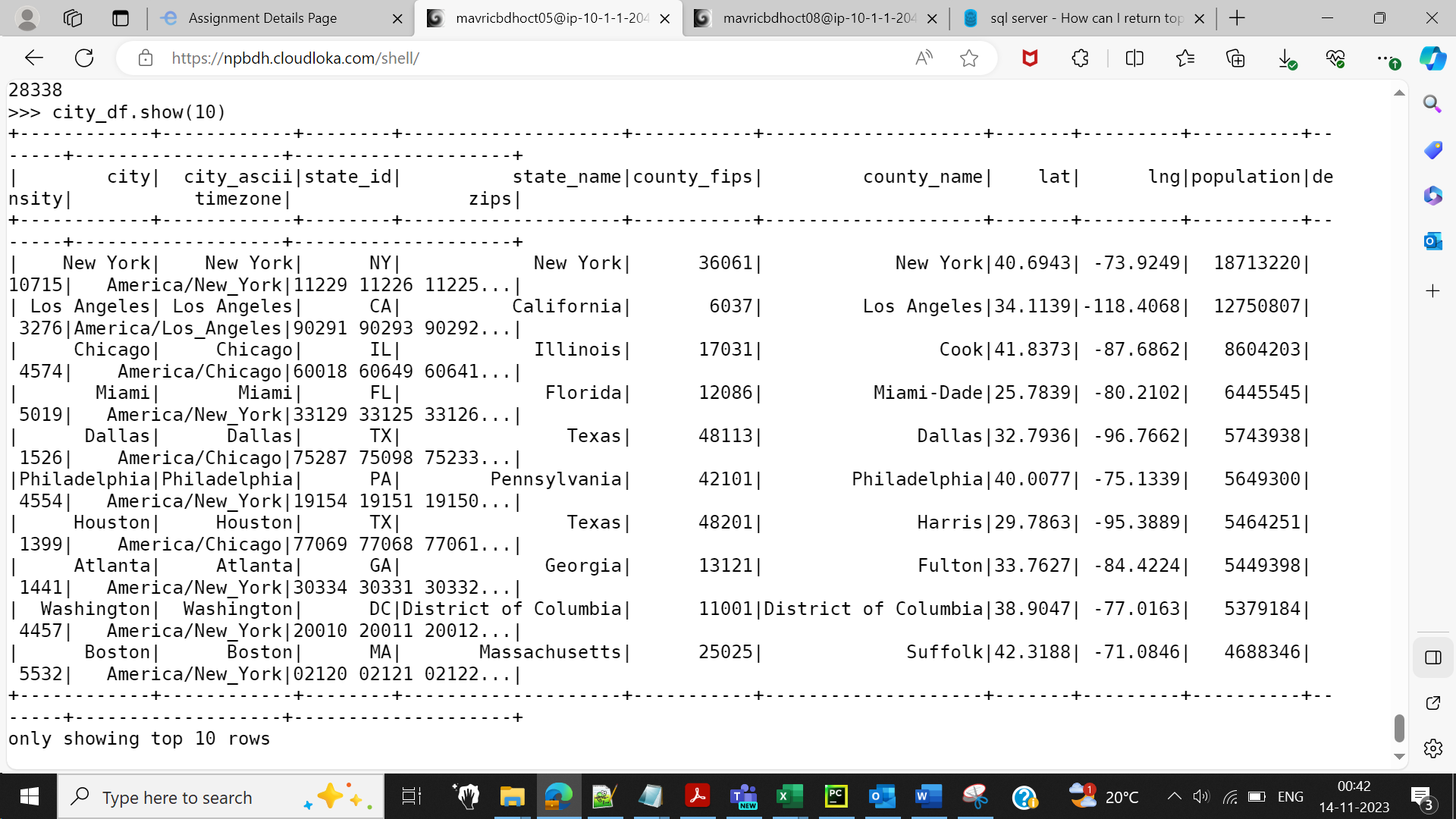
|-- zips: string (nullable = true)

A screenshot of a computer code

Description automatically generated

city\_df.count()



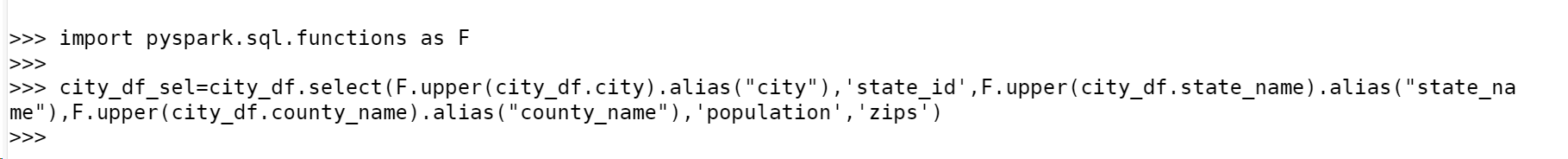


Selecting only required Columns in city data file like city, state\_id,state\_name,county\_name,population,zips

Also converting city, state, county fields to upper case.

import pyspark.sql.functions as F

city\_df\_sel=city\_df.select(F.upper(city\_df.city).alias("city"),'state\_id',F.upper(city\_df.state\_name).alias("state\_name"),F.upper(city\_df.county\_name).alias("county\_name"),'population','zips')



>>> city\_df\_sel.printSchema()

root

|-- city: string (nullable = true)

|-- state\_id: string (nullable = true)

|-- state\_name: string (nullable = true)

|-- county\_name: string (nullable = true)

|-- population: integer (nullable = true)

|-- zips: string (nullable = true)

A computer screen shot of a code

Description automatically generated

>>> city\_df\_sel.show(4)

+-----------+--------+----------+-----------+----------+--------------------+

| city|state\_id|state\_name|county\_name|population| zips|

+-----------+--------+----------+-----------+----------+--------------------+

| NEW YORK| NY| NEW YORK| NEW YORK| 18713220|11229 11226 11225...|

|LOS ANGELES| CA|CALIFORNIA|LOS ANGELES| 12750807|90291 90293 90292...|

| CHICAGO| IL| ILLINOIS| COOK| 8604203|60018 60649 60641...|

| MIAMI| FL| FLORIDA| MIAMI-DADE| 6445545|33129 33125 33126...|

+-----------+--------+----------+-----------+----------+--------------------+

A screenshot of a computer

Description automatically generated

**Clean Prescriber Data:**

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

A computer screen shot of a computer screen

Description automatically generated

prescriber\_df.show(10)

A close-up of a computer screen

Description automatically generated

prescriber\_df.printSchema()

A computer screen shot of a code

Description automatically generated

A computer screen shot of a code

Description automatically generated

prescriber\_df.count()

>>> prescriber\_df.count()

1329329

>>>

A white background with black text

Description automatically generated

**Step1. Selecting only required column and renaming the fields to shorter names.**

prescriber\_df\_sel=prescriber\_df.select(prescriber\_df.npi.alias("presc\_id"),(prescriber\_df.nppes\_provider\_last\_org\_name).alias ('presc\_lname'),(prescriber\_df.nppes\_provider\_first\_name).alias ('presc\_fname'),

(prescriber\_df.nppes\_provider\_city).alias ('presc\_city'),(prescriber\_df.nppes\_provider\_state).alias ('presc\_state'),(prescriber\_df.specialty\_description).alias ('presc\_spclt'),

(prescriber\_df.drug\_name).alias ('drug\_nm'),(prescriber\_df.total\_claim\_count).alias ('tot\_clm\_count'),(prescriber\_df.total\_day\_supply).alias ('tot\_day\_supp'),

(prescriber\_df.total\_drug\_cost) .alias ('tot\_drug\_cst'),prescriber\_df.years\_of\_exp)

A screenshot of a computer

Description automatically generated

prescriber\_df\_sel.printSchema()

A screenshot of a computer code

Description automatically generated

Step2. Add a Country Field 'USA'

from pyspark.sql.functions import col, lit

prescriber\_df\_sel=prescriber\_df\_sel.withColumn("Country", lit("USA"))

A close-up of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

prescriber\_df\_sel.show(10)

A screenshot of a computer

Description automatically generated

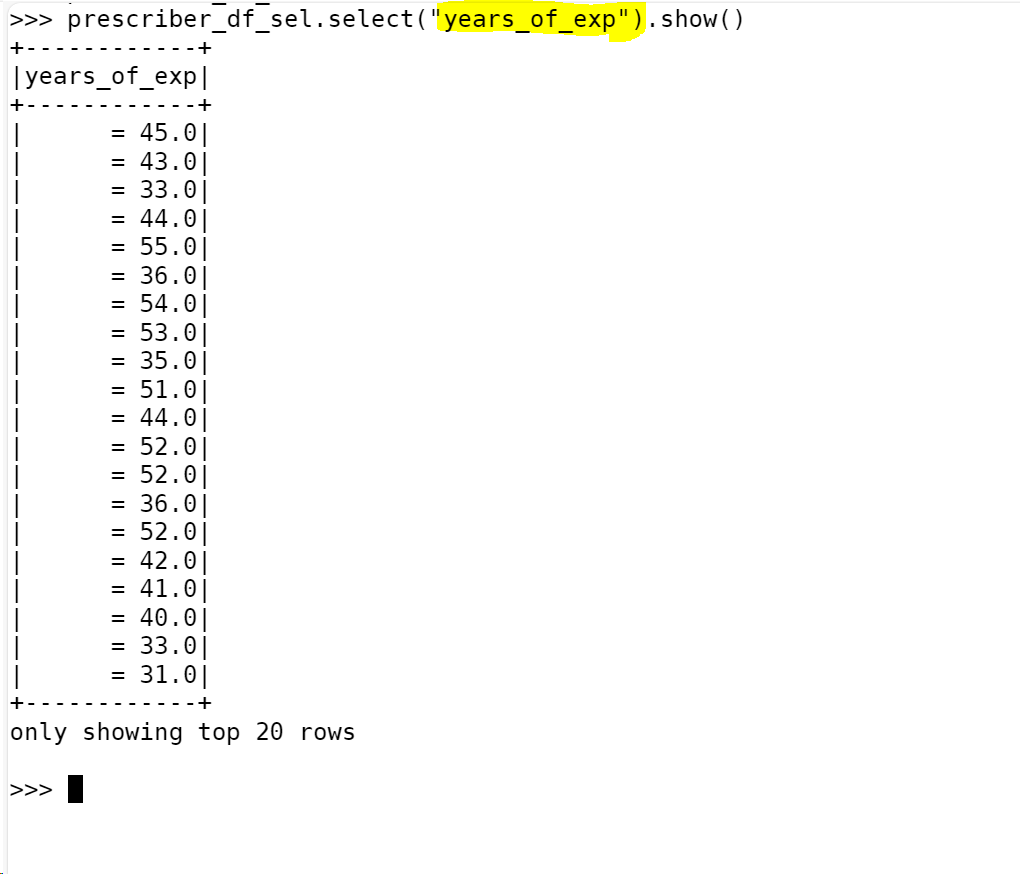
**Step3: Clean years\_of\_exp field**

import pyspark.sql.functions as F

prescriber\_df\_sel = prescriber\_df\_sel.withColumn("years\_of\_exp",F.regexp\_extract(col("years\_of\_exp"),'\d+',0))

prescriber\_df\_sel.select("years\_of\_exp").s

how()



A close-up of a text

Description automatically generated

prescriber\_df\_sel.select("years\_of\_exp").show()

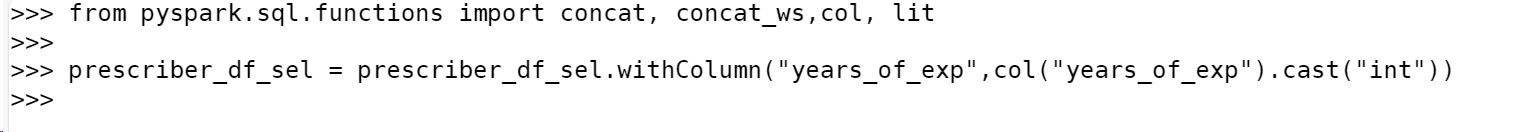
A screenshot of a computer

Description automatically generated

**Step4: Convert the years\_of\_exp datatype from string to Number**

from pyspark.sql.functions import concat, concat\_ws,col, lit

prescriber\_df\_sel = prescriber\_df\_sel.withColumn("years\_of\_exp",col("years\_of\_exp").cast("int"))



prescriber\_df\_sel.printSchema()

A computer screen shot of a code

Description automatically generated

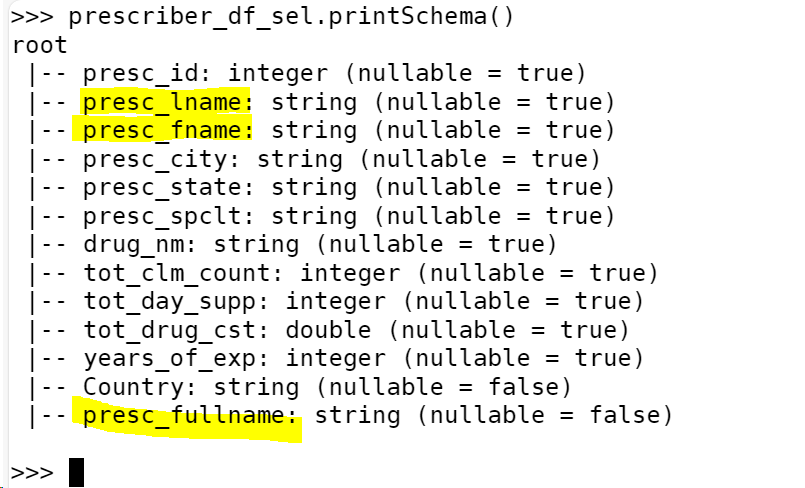
**Step5: Combine First Name and Last Name and remove the individual column**

from pyspark.sql.functions import concat, concat\_ws,col, lit

prescriber\_df\_sel = prescriber\_df\_sel.withColumn("presc\_fullname",concat\_ws(" ", "presc\_fname", "presc\_lname"))

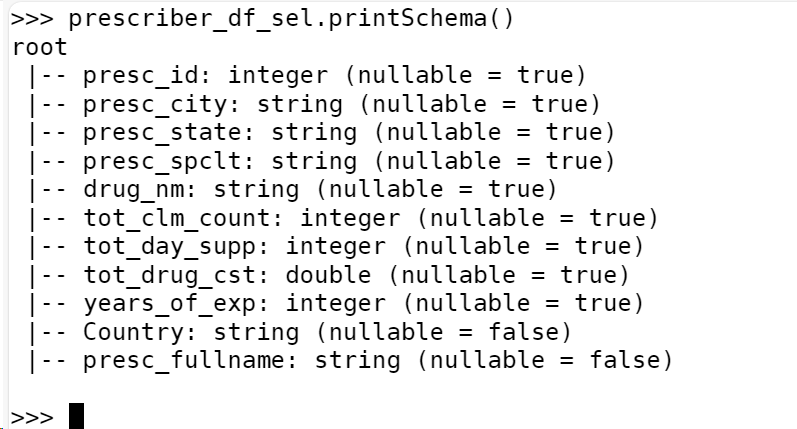
A close up of a white background

Description automatically generated



Removing presc\_fname and presc\_lname

prescriber\_df\_sel = prescriber\_df\_sel.drop("presc\_fname", "presc\_lname")



**Step6: Count the number of null values for each column**

from pyspark.sql.functions import col,isnan, when, count

prescriber\_df\_sel.select([count(F.when(isnan(c) | col(c).isNull(),c)).alias(c) for c in

prescriber\_df\_sel.columns]).show()

A white background with black lines and dots

Description automatically generated

+--------+----------+-----------+-----------+-------+-------------+------------+------------+------------+-------+-------

-------+

|presc\_id|presc\_city|presc\_state|presc\_spclt|drug\_nm|tot\_clm\_count|tot\_day\_supp|tot\_drug\_cst|years\_of\_exp|Country|presc\_f

ullname|

+--------+----------+-----------+-----------+-------+-------------+------------+------------+------------+-------+-------

-------+

| 22| 1| 1| 1| 15| 3| 1| 1| 1| 0|

0|

+--------+----------+-----------+-----------+-------+-------------+------------+------------+------------+-------+-------

-------+

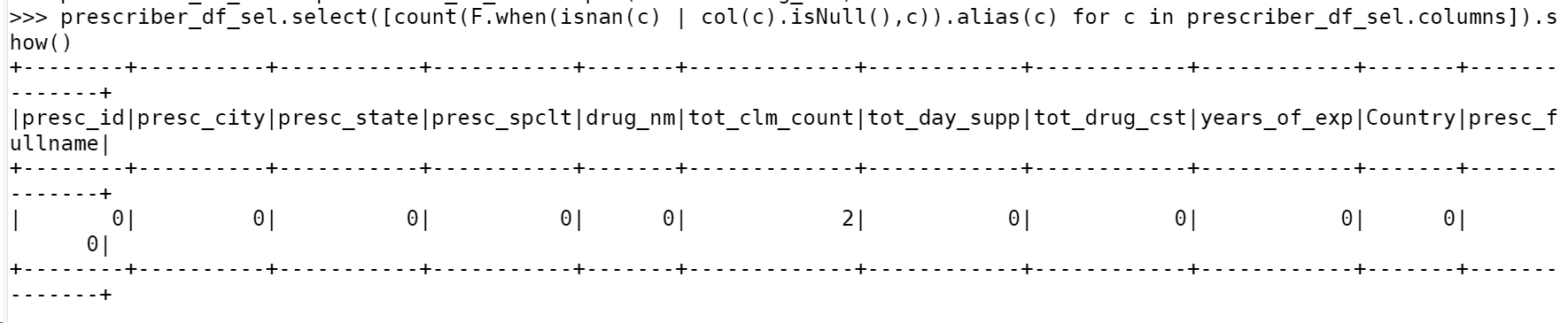
**Step7: clean all the Null/Nan Values where presc\_id and drug\_nm is null**

prescriber\_df\_sel = prescriber\_df\_sel.dropna(subset="presc\_id")

prescriber\_df\_sel = prescriber\_df\_sel.dropna(subset="drug\_nm")

A computer screen shot of a computer code

Description automatically generated



**DATA TRANSFORMATION**

Prescriber Report:

* 1. Top 5 Prescribers with highest total\_claim\_count per each state.
  2. Consider the prescribers only from 20 to 50 years of experience.

from pyspark.sql import Window

from pyspark.sql.functions import dense\_rank

from pyspark.sql.functions import concat, concat\_ws,col, lit

window = Window.partitionBy("presc\_state").orderBy(col("tot\_clm\_count").desc())

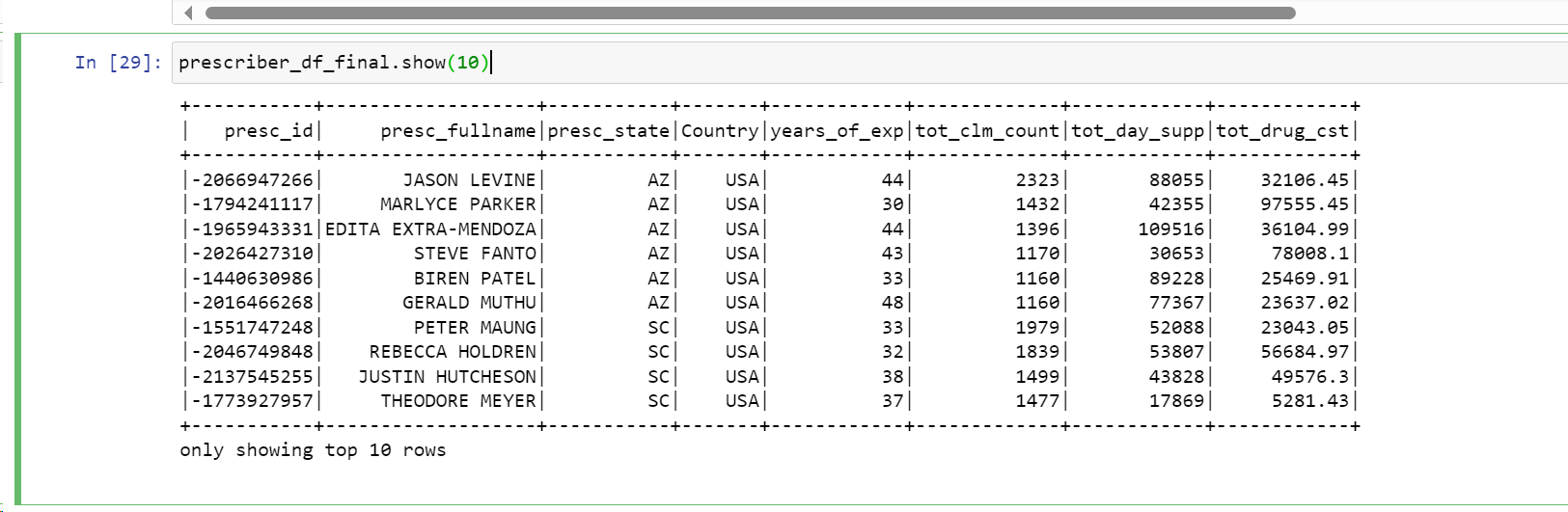
prescriber\_df\_final = prescriber\_df\_sel.select("presc\_id","presc\_fullname","presc\_state","Country","years\_of\_exp","tot\_clm\_count","tot\_day\_supp","tot\_drug\_cst") \

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

.withColumn("dense\_rank",dense\_rank().over(window)) \

.filter(col("dense\_rank") <= 5) \

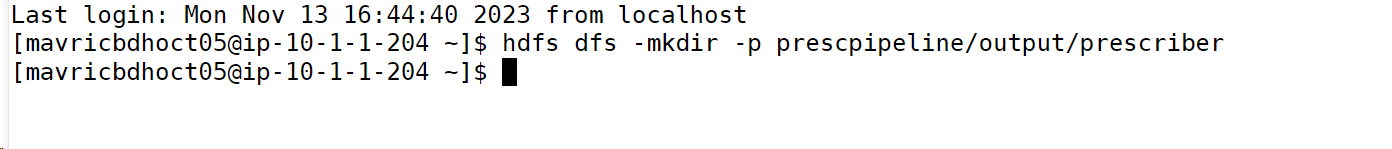
.select("presc\_id","presc\_fullname","presc\_state","Country","years\_of\_exp","tot\_clm\_count","tot\_day\_supp","tot\_drug\_cst")



Creating Output directory :

\*\*\*\*Run in HDFS\*\*\*

hdfs dfs -mkdir -p prescpipeline/output/prescriber

 A computer screen with text

Description automatically generated

**Writing the output into HDFS in ORC format and snappy compression :**

from pyspark.sql import SparkSession

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

spark.sparkContext.setLogLevel("ERROR")

**# writing the output in HDFS with split 2**

prescriber\_df\_final.repartition(2).write.mode("overwrite").option("compression","snappy").orc("/user/mavricbdhoct05/prescpipeline/output/prescriber/prescriber\_report.orc")

**checking the output report file generated in hdfs:**

hdfs dfs -ls /user/mavericbdhoct05/prescpipeline/output/prescriber/prescriber\_report.orc/

A black screen with white text

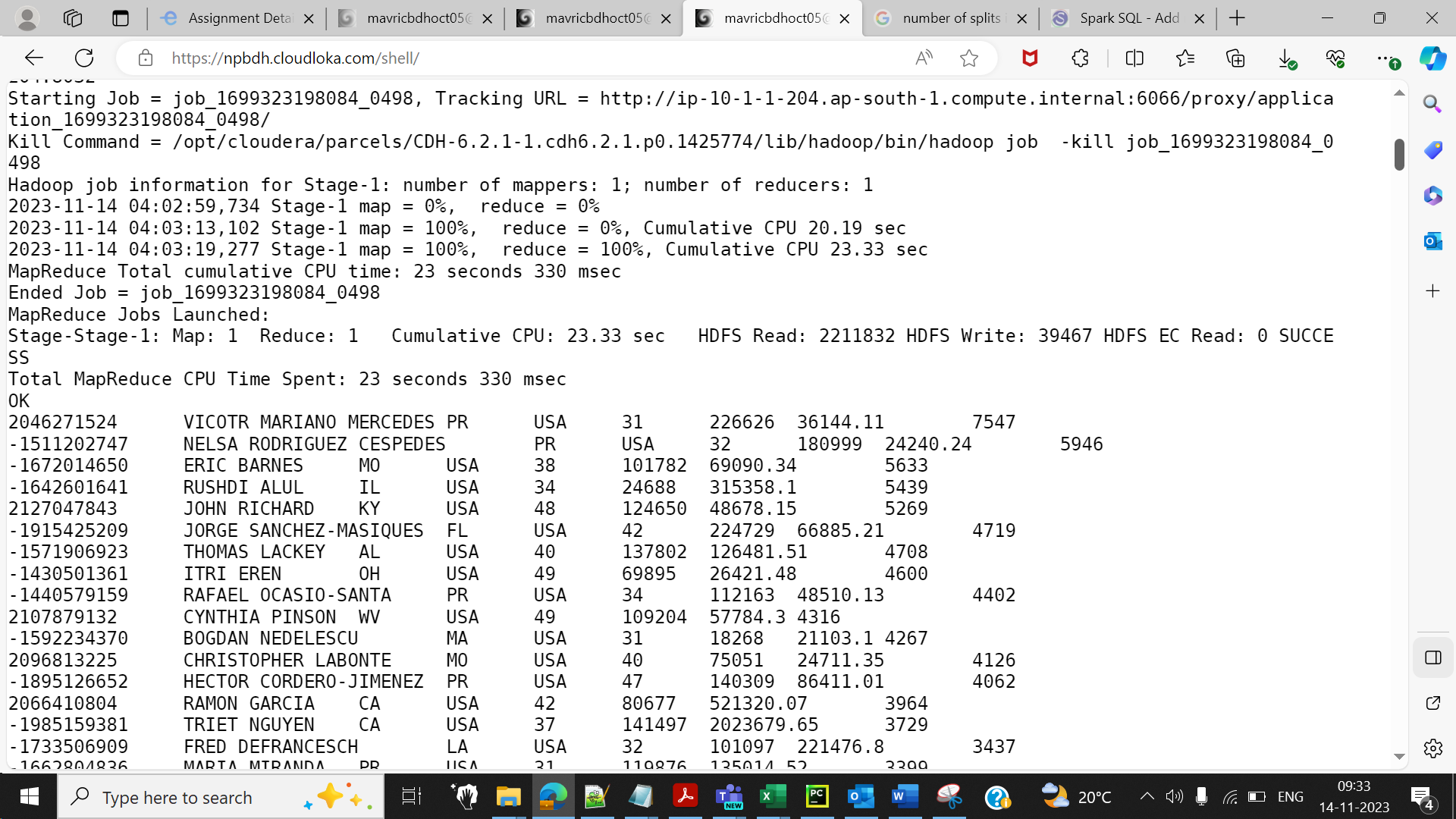
Description automatically generated

prescriber\_report\_DF.write.mode("overwrite").saveAsTable("aastha\_lab. prescriber\_report\_DF ")

use aastha\_lab;

show tables;

select \* from prescriber\_report\_DF order by Total\_claim\_count desc ;



* 1. **Transform Logic: City Report**
  2. **1.Calculate the Number of zips in each city.**

from pyspark.sql.functions import split,size

df\_city\_split = city\_df\_sel.withColumn('zip\_counts',size(split(city\_df\_sel['zips']," ")))

df\_city\_split.show(5)

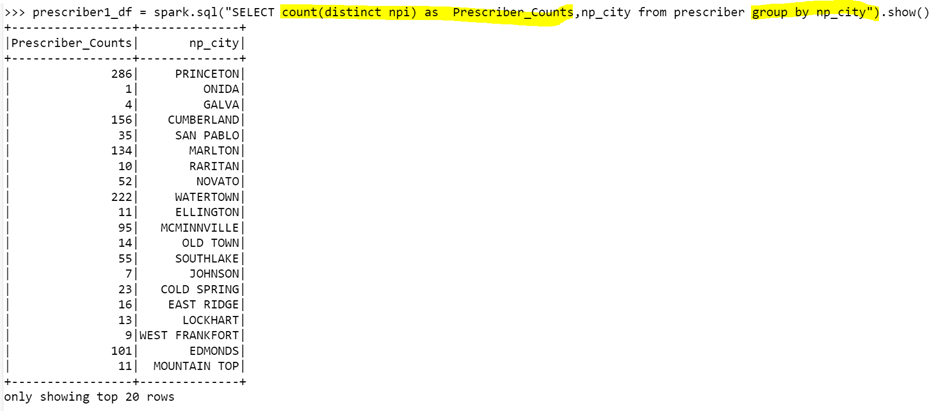
A screenshot of a computer

Description automatically generated

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

prescriber\_df\_sel.createOrReplaceTempView("prescriber")

prescriber1\_df = spark.sql("SELECT count(distinct presc\_id) as presc\_counts,presc\_city from prescriber group by presc\_city").show()



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

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

A screenshot of a computer

Description automatically generated

prescriber1\_df.createOrReplaceTempView(“prescriber1”)

A screenshot of a computer

Description automatically generated

df\_city\_split.createOrReplaceTempView(“City\_split\_df”)

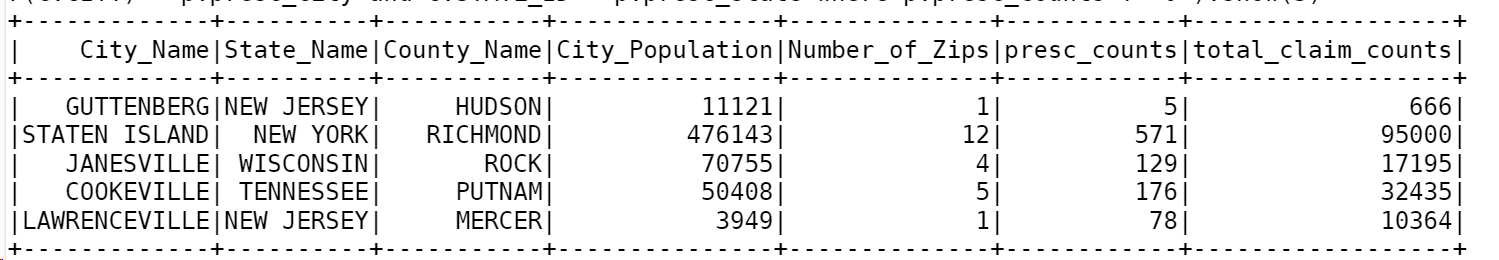
A screenshot of a computer

Description automatically generated

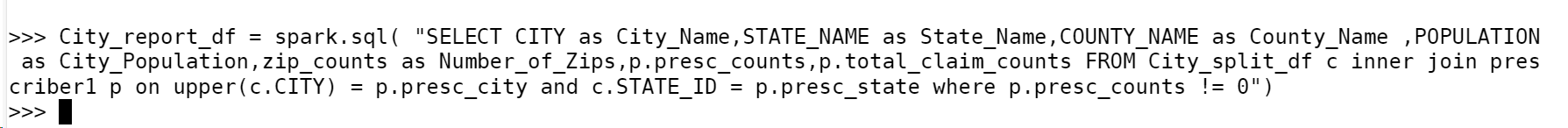
**Do not report a city in the final report if no prescriber is assigned to it.**

# **Joining two tables prescriber and1 and prescriber1**

spark.sql( "SELECT CITY as City\_Name,STATE\_NAME as State\_Name,COUNTY\_NAME as County\_Name ,POPULATION as City\_Population,zip\_counts as Number\_of\_Zips,p.presc\_counts,p.total\_claim\_counts FROM City\_split\_df c inner join prescriber1 p on upper(c.CITY) = p.presc\_city and c.STATE\_ID = p.presc\_state where p.presc\_counts != 0").show(5)



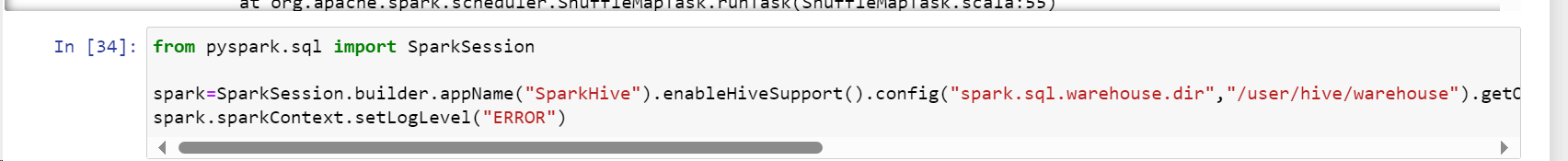
City\_report\_df = spark.sql( "SELECT CITY as City\_Name,STATE\_NAME as State\_Name,COUNTY\_NAME as County\_Name ,POPULATION as City\_Population,zip\_counts as Number\_of\_Zips,p.presc\_counts,p.total\_claim\_counts FROM City\_split\_df c inner join prescriber1 p on upper(c.CITY) = p.presc\_city and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



from pyspark.sql import SparkSession

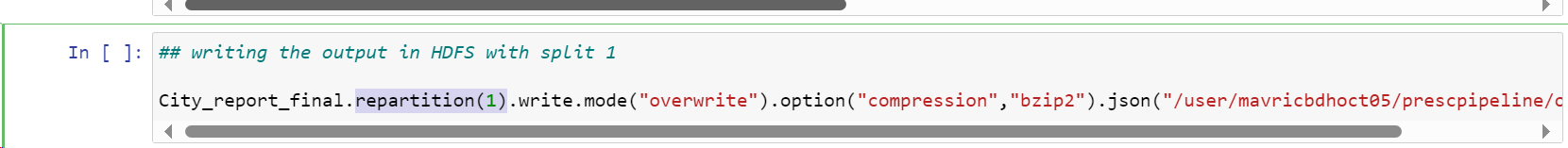
spark=SparkSession.builder.appName("SparkHive").enableHiveSupport().config("spark.sql.warehouse.dir","/user/hive/warehouse").getOrCreate()

spark.sparkContext.setLogLevel("ERROR")



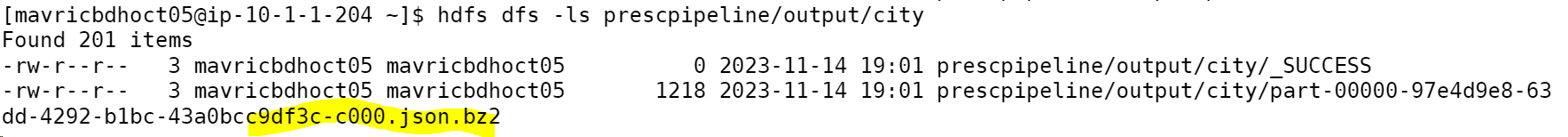
#Storing the output into HDFS

City\_report\_df.write. repartition(1) mode("overwrite").option("compression","bzip2").json("/user/mavricbdhoct05/prescpipeline/output/city")



City output In HDFS

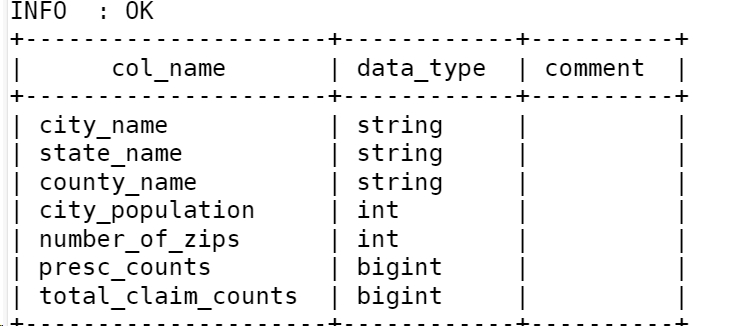
hdfs dfs -ls /user/mavricbdhoct05/prescpipeline/output/city



City\_report\_df.write.mode("overwrite").saveAsTable("aastha\_lab.City\_Final\_Report")

desc City\_Final\_Report;

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



A screenshot of a computer

Description automatically generated

A close-up of a document

Description automatically generated