**Project: Health care Data Analysis**

**US Prescriber report**

**Data Ingestion**

Creating input Directory: prescpipeline/staging/prescriber and placed the input file into hdfs input folder

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

[maverickavitha@ip-10-1-1-204 ~]$ hdfs dfs -put USA\_Presc\_Medicare\_Data\_12021.csv prescpipeline/staging/prescriber

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

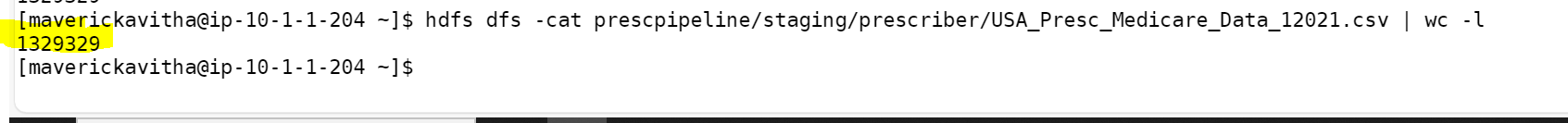


A screenshot of a computer

Description automatically generated

Checking the count in source file :

hdfs dfs -cat prescpipeline/staging/prescriber/USA\_Presc\_Medicare\_Data\_12021.csv | wc -l



**Preprocessing / Data Cleansing:**

>>> prescriberDF=spark.read.option("header",True).option("inferschema",True).csv("prescpipeline/staging/prescriber/USA\_Presc\_Medicare\_D

ata\_12021.csv")

>>> prescriberDF.show(10)

A close up of a document

Description automatically generated

>>> prescriberDF.printSchema()

A computer screen shot of a code

Description automatically generated

>>> prescriberDF.count()

A screenshot of a computer

Description automatically generated

Data Cleansing (Clean Prescriber Data ) :

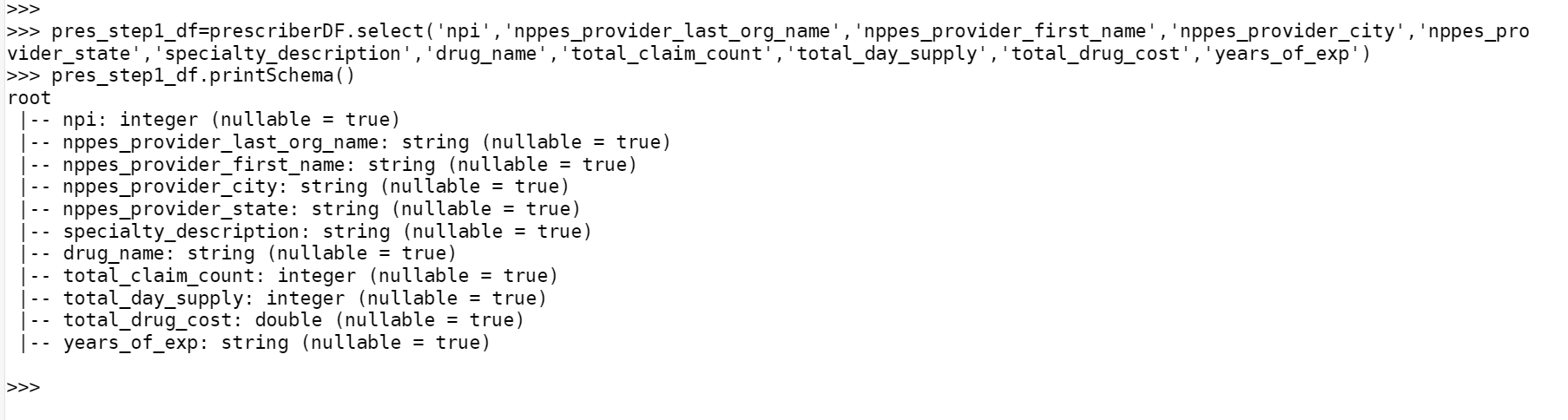
Step 1:

Select only required fields

>>> pres\_step1\_df=prescriberDF.select('npi','nppes\_provider\_last\_org\_name','nppes\_provider\_first\_name','nppes\_provider\_city','nppes\_pro

vider\_state','specialty\_description','drug\_name','total\_claim\_count','total\_day\_supply','total\_drug\_cost','years\_of\_exp')

>>> pres\_step1\_df.printSchema()



Step 2:

Rename the above fields to shorter names :

>>> pres\_step1\_df=prescriberDF.select('npi',(prescriberDF.nppes\_provider\_last\_org\_name).alias ('np\_Lname'),(prescriberDF.nppes\_provider

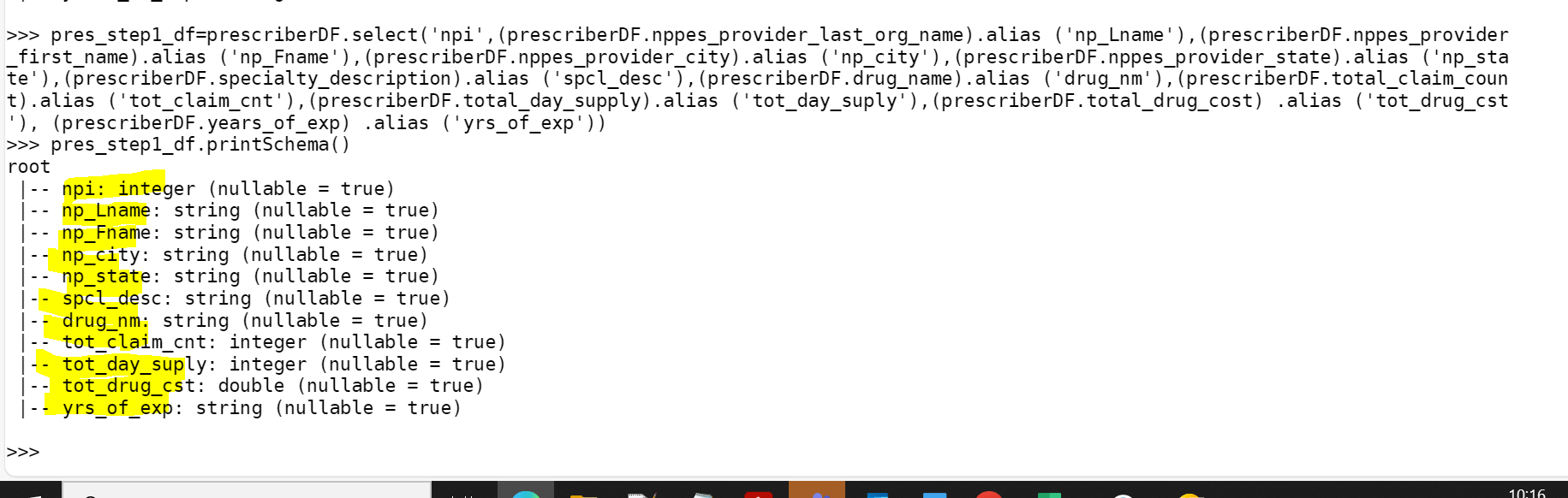
\_first\_name).alias ('np\_Fname'),(prescriberDF.nppes\_provider\_city).alias ('np\_city'),(prescriberDF.nppes\_provider\_state).alias ('np\_sta

te'),(prescriberDF.specialty\_description).alias ('spcl\_desc'),(prescriberDF.drug\_name).alias ('drug\_nm'),(prescriberDF.total\_claim\_coun

t).alias ('tot\_claim\_cnt'),(prescriberDF.total\_day\_supply).alias ('tot\_day\_suply'),(prescriberDF.total\_drug\_cost) .alias ('tot\_drug\_cst

'), (prescriberDF.years\_of\_exp) .alias ('yrs\_of\_exp'))

>>> pres\_step1\_df.printSchema()



Step 3:

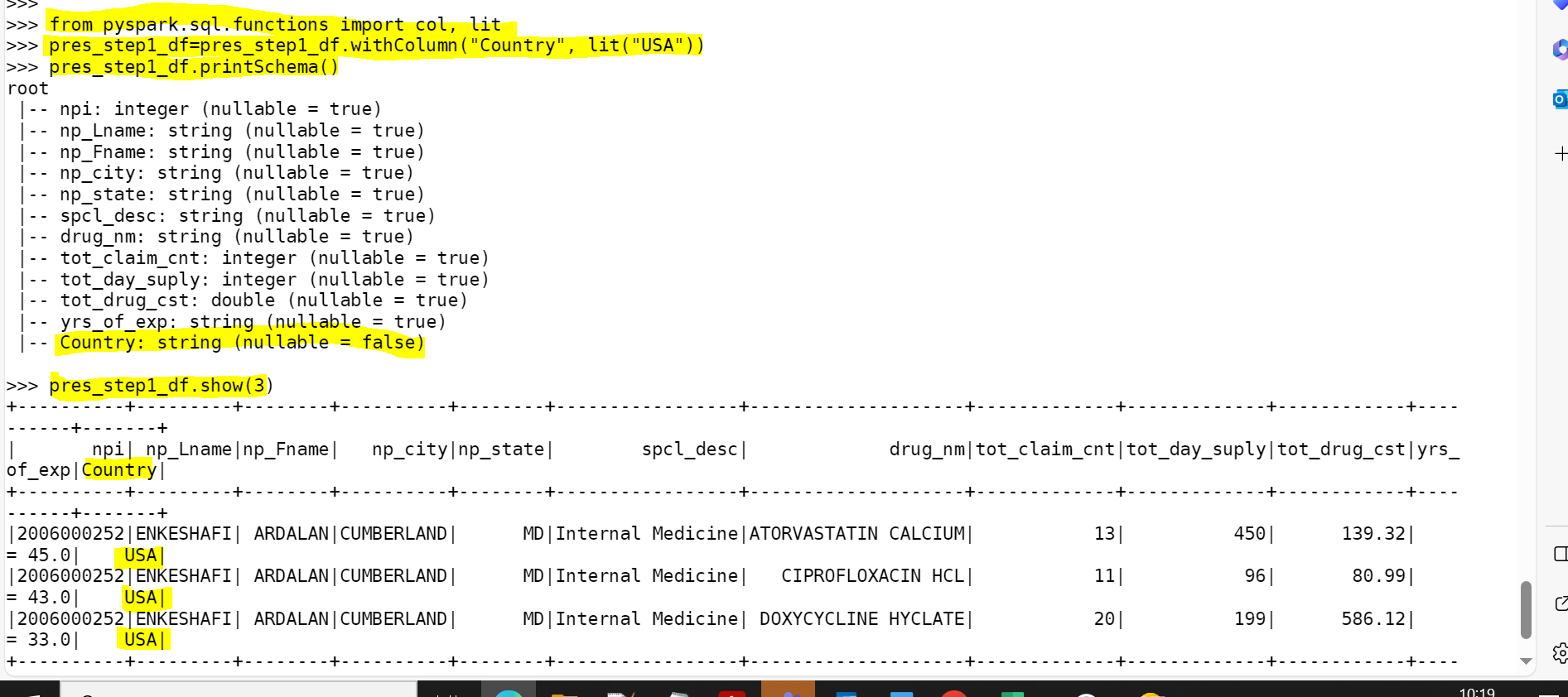
Add a Country Field 'USA' to the above data

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

>>> pres\_step1\_df=pres\_step1\_df.withColumn("Country", lit("USA"))

>>> pres\_step1\_df.printSchema()

>>> pres\_step1\_df.show(3)



Step 4:

Clean the “years\_of\_exp” to extract only the numbers.

* 1. Hint : use regexp\_extract from the package pyspark.sql.functions

>>> import pyspark.sql.functions as f

>>> pres\_step1\_df = pres\_step1\_df.withColumn("yrs\_of\_exp", f.regexp\_extract("yrs\_of\_exp", r'([0-9]+)', 1))

A screenshot of a computer

Description automatically generated

Step 5:

Convert the years\_of\_exp field to integer

>>> import pyspark.sql.functions as f

>>> pres\_step1\_df= pres\_step1\_df.na.fill(value=0,subset=["yrs\_of\_exp"])

>>> from pyspark.sql.types import IntegerType

>>> pres\_step1\_df = pres\_step1\_df.withColumn("yrs\_of\_exp", pres\_step1\_df["yrs\_of\_exp"].cast(IntegerType()))

>>> pres\_step1\_df.printSchema()

A screenshot of a computer

Description automatically generated

Step 6:

Combine First Name and Last Name in to a single field and remove the individual columns

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

>>> pres\_step1\_df = pres\_step1\_df.withColumn("prsc\_fullname",concat\_ws(" ", "np\_Fname", "np\_Lname"))

>>> pres\_step1\_df.show(3)

A screenshot of a computer

Description automatically generated

Dropping individual columns:

>>> pres\_step1\_df = pres\_step1\_df.drop("np\_Fname", "np\_Lname")

>>> pres\_step1\_df.printSchema()

>>> pres\_step1\_df.show(3)

A screenshot of a computer

Description automatically generated

Step 7:

Count the number of null values for each column

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

>>> pres\_step1\_df=pres\_step1\_df.select([count(f.when(isnan(c) | col(c).isNull(),c)).alias(c) for c in pres\_step1\_df.columns]).show()

A screen shot of a computer

Description automatically generated

Step 8:

clean all the Null/Nan Values

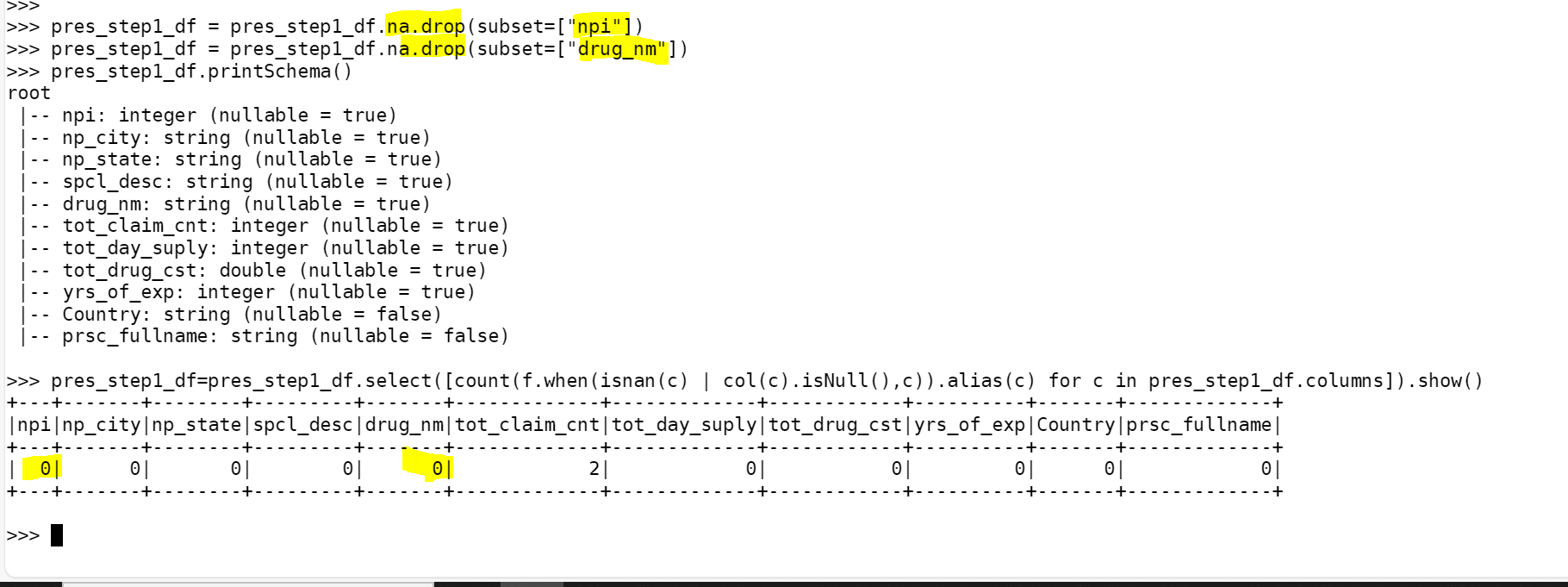
Delete the records where the PRESC\_ID and DRUG\_NAME is fields are NULL. Use dropna() of dataframe

>>> pres\_step1\_df = pres\_step1\_df.na.drop(subset=["npi"])

>>> pres\_step1\_df = pres\_step1\_df.na.drop(subset=["drug\_nm"])

>>> pres\_step1\_df.printSchema()

>>> pres\_step1\_df=pres\_step1\_df.select([count(f.when(isnan(c) | col(c).isNull(),c)).alias(c) for c in pres\_step1\_df.columns]).show()



**Transformation:**

Step 1:

Filtering the prescribers only from 20 to 50 years of experience.

* 1. >>> from pyspark.sql import functions as F

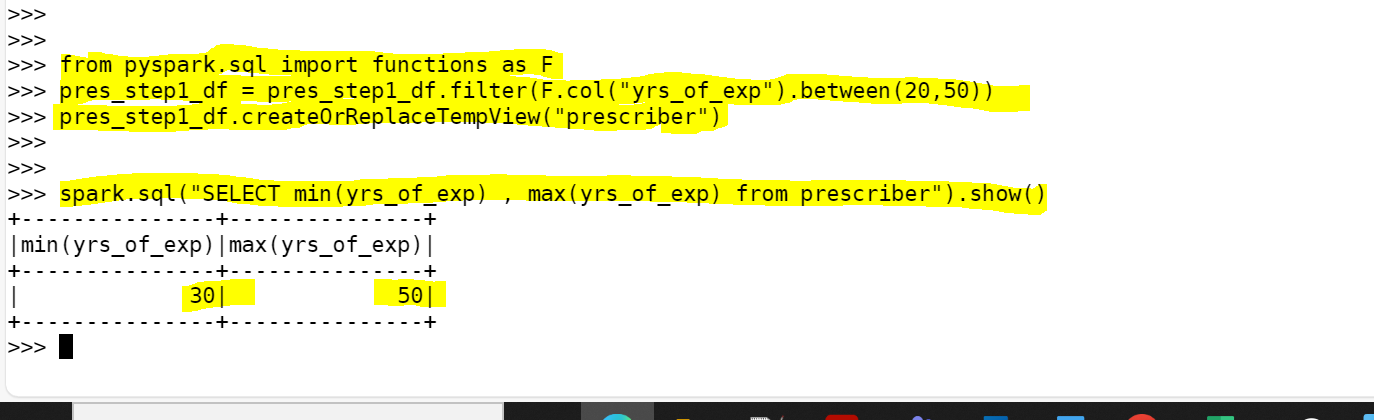
>>> pres\_step1\_df = pres\_step1\_df.filter(F.col("yrs\_of\_exp").between(20,50))

>>> pres\_step1\_df.createOrReplaceTempView("prescriber")

>>>

>>>

>>> spark.sql("SELECT min(yrs\_of\_exp) , max(yrs\_of\_exp) from prescriber").show()



Step 2:

Selecting Top 5 Prescribers with highest total\_claim\_count per each state.

>>> 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("np\_state").orderBy(col("tot\_claim\_cnt").desc())

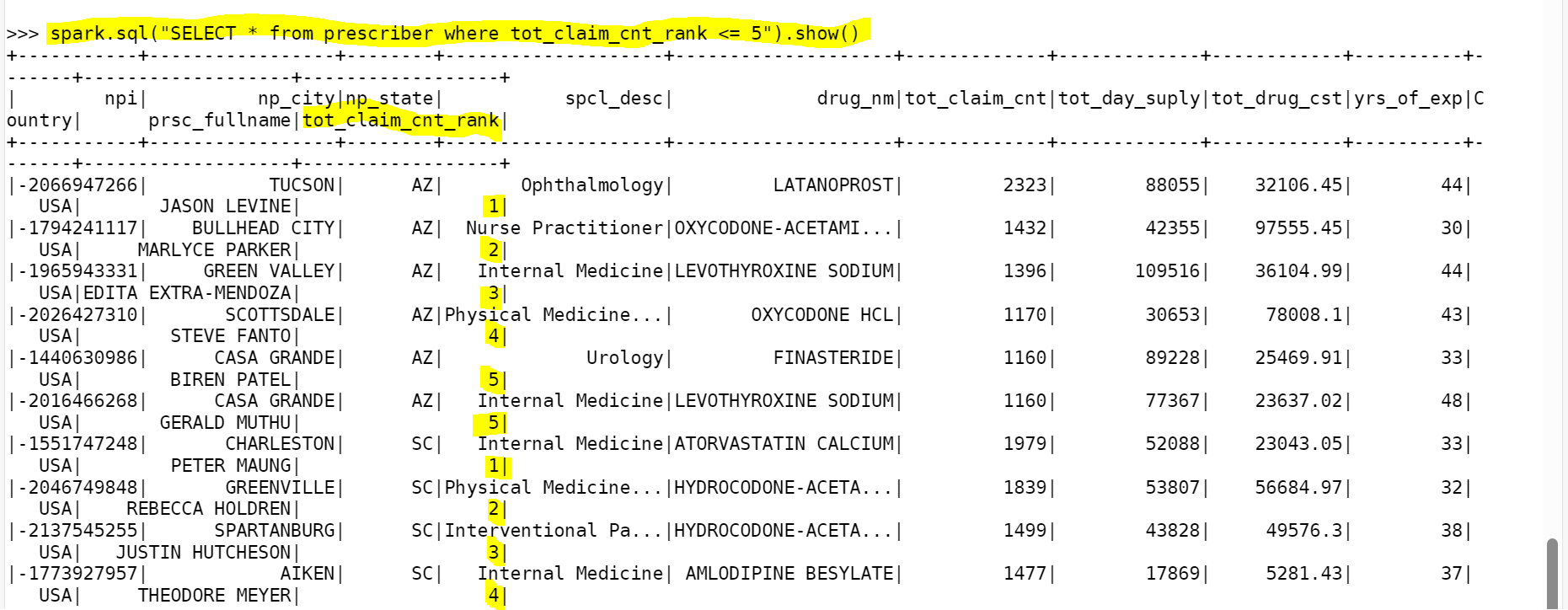
>>> pres\_step1\_df = pres\_step1\_df.withColumn("tot\_claim\_cnt\_rank", dense\_rank().over(window))

>>> pres\_step1\_df.createOrReplaceTempView("prescriber")

A screenshot of a computer

Description automatically generated

>>> spark.sql("SELECT \* from prescriber where tot\_claim\_cnt\_rank <= 5").show()



Checking count in final report :

>>> spark.sql("SELECT count(\*) from prescriber where tot\_claim\_cnt\_rank <= 5").show()

A white background with black text

Description automatically generated

Selecting the specific columns required for Final report :

>>> spark.sql("SELECT npi as Prescriber\_ID , prsc\_fullname as Prescriber\_Full\_Name, np\_state as Prescriber\_State,Country as Prescriber

\_Country, yrs\_of\_exp as Prescriber\_Years\_of\_Experience,tot\_claim\_cnt as Total\_claim\_count,tot\_day\_suply as Total\_Days\_Supply,tot\_drug\_c

st as Total\_Drug\_Cost from prescriber where tot\_claim\_cnt\_rank <= 5").show()

A screenshot of a computer

Description automatically generated

Report Layout:

A screenshot of a computer

Description automatically generated

Writing the output/ Final report in HDFS and Hive

Creating Output Directory:

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

A yellow and black text

Description automatically generated

Writing data in .ORC format and snappy compression :

>>> from pyspark.sql import SparkSession

>>> spark=SparkSession.builder.appName("Spark Hive").enableHiveSupport().config("spark.sql.warehouse.dir","/user/hive/warehouse").getOr

Create()

>>> spark.sparkContext.setLogLevel("ERROR")

>>> prescriber\_report\_DF = spark.sql("SELECT npi as Prescriber\_ID , prsc\_fullname as Prescriber\_Full\_Name, np\_state as Prescriber\_Stat

e,Country as Prescriber\_Country, yrs\_of\_exp as Prescriber\_Years\_of\_Experience,tot\_claim\_cnt as Total\_claim\_count,tot\_day\_suply as Total

\_Days\_Supply,tot\_drug\_cst as Total\_Drug\_Cost from prescriber where tot\_claim\_cnt\_rank <= 5")

>>>

>>> repart\_df = prescriber\_report\_DF.repartition(2)

>>>

>>> repart\_df.write.mode("overwrite").option("compression","snappy").orc("/user/maverickavitha/prescpipeline/output/prescriber")

A screenshot of a computer

Description automatically generated

Report generated in HDFS:

hdfs dfs -ls /user/maverickavitha/prescpipeline/output/prescriber/

A screenshot of a computer

Description automatically generated

Writing the data into Hive

>>> repart\_df.write.partitionBy("Total\_claim\_count").mode("overwrite").saveAsTable("Final\_Prescriber\_Report")

A screenshot of a computer

Description automatically generated

Data in hive

A close-up of a document

Description automatically generated

Count in Hive :

A yellow line on a white paper

Description automatically generated