**Big Data Capstone Project : Health care Data Analysis**

**Project Flow:**

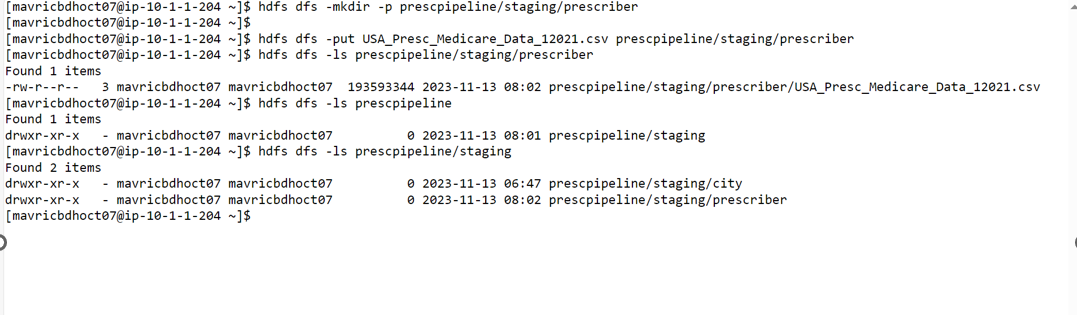
1. **Prescriber\_report\_capstone\_project:**

**Data Ingestion:**

**step1:** Sample data files are provided and they have to be ingested in to HDFS in to the project’s location

**hdfs dfs -mkdir -p prescpipeline/staging/prescriber**

**hdfs dfs -put USA\_Presc\_Medicare\_Data\_12021.csv prescpipeline/staging/prescriber**



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

**prescriberDF.show(10)**

A screenshot of a computer

Description automatically generated

**prescriberDF.printSchema()**

**prescriberDF.count()**

A white background with black text

Description automatically generated

**Data Cleansing/Preprocessing:**

**Selecting the required columns**

**PrescriberDF1=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')**

A blue and white text on a white background

Description automatically generated

**Rename the above fields to shorter names**

**PrescriberDF1=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'))**

A computer screen shot of a blue line

Description automatically generated

**Add a Country Field 'USA' to the above data**

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

**PrescriberDF1=PrescriberDF1.withColumn("Country", lit("USA"))**

**PrescriberDF1.show(3)**

A screenshot of a computer

Description automatically generated

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

(Hint : use regexp\_extract from the package pyspark.sql.functions)

**Adding years of exp Column to PrescriberDF1:**

**PrescriberDF1=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')**

A white background with black text

Description automatically generated

**Removing special Characters from years of exp column:**

**import pyspark.sql.functions as F**

**PrescriberDF1= PrescriberDF1.withColumn("years\_of\_exp", F.regexp\_replace("years\_of\_exp", "= ", ""))**

**PrescriberDF1.show(10)**

A screenshot of a computer

Description automatically generated

**Convert the years\_of\_exp field to integer**

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

PrescriberDF2 = PrescriberDF1.withColumn("years\_of\_exp",col("years\_of\_exp").cast("int"))

A screenshot of a computer

Description automatically generated

PrescriberDF1.show()

A screenshot of a computer

Description automatically generated

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

**Combining columns:**

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

**PrescriberDF2 = PrescriberDF1.withColumn("presc\_fullname",concat\_ws(" ", "np\_Fname", "np\_Lname"))**

**PrescriberDF1.show(3)**

A screenshot of a computer

Description automatically generated

**Removing individual columns**

**PrescriberDF1 = PrescriberDF1.drop("np\_Fname", "np\_Lname")**

**PrescriberDF1.show(3)**

A close-up of a document

Description automatically generated

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

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

**PrescriberDF1=PrescriberDF1.select([count(F.when(isnan(c) | col(c).isNull(),c)).alias(c) for c in PrescriberDF1.columns]).show()**

A screenshot of a computer program

Description automatically generated

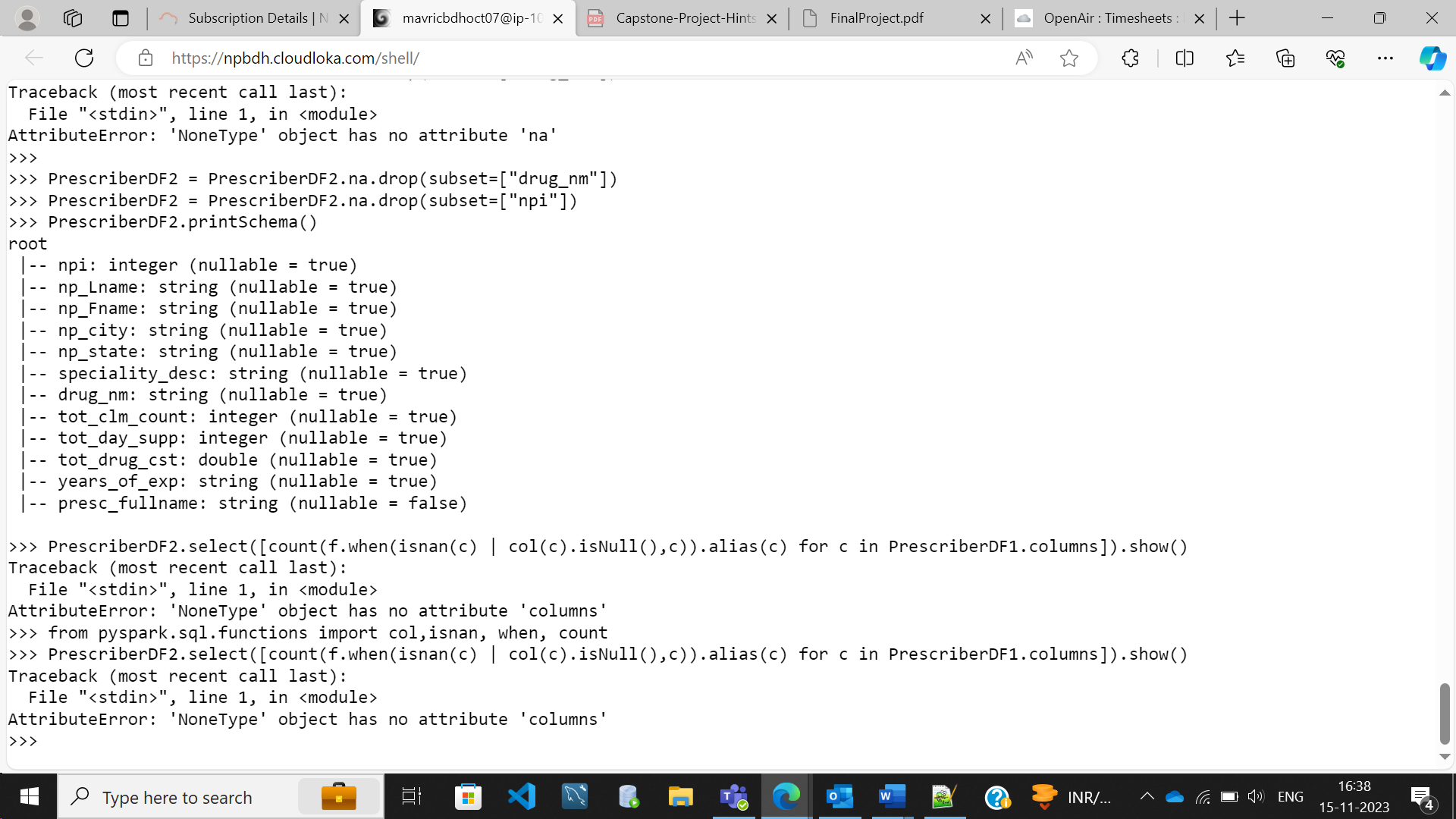
**Clean all the Null/Nan Values**

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

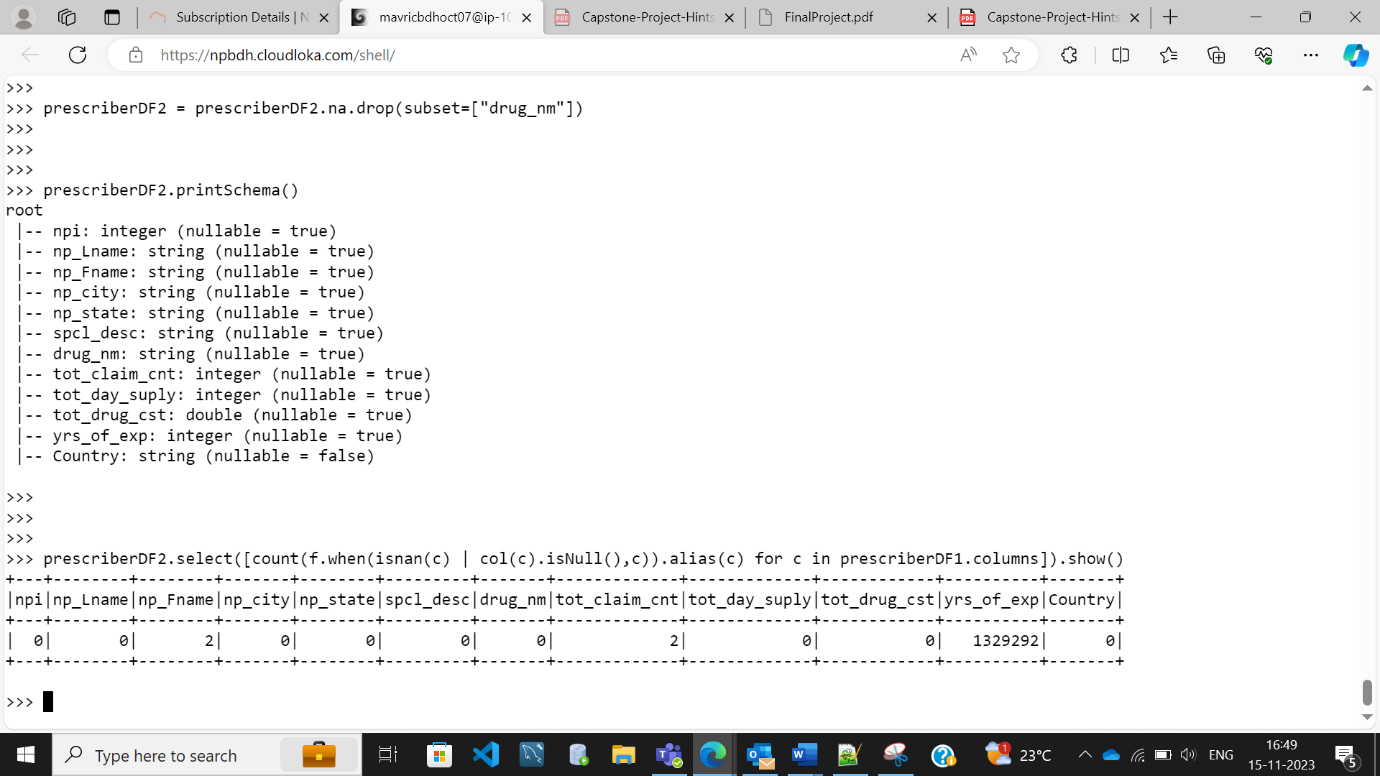
**PrescriberDF2 = PrescriberDF1.na.drop(subset=["npi"])**

**PrescriberDF2 = PrescriberDF1.na.drop(subset=["drug\_nm"])**

**PrescriberDF2.printSchema()**



**PrescriberDF2.select([count(f.when(isnan(c) | col(c).isNull(),c)).alias(c) for c in PrescriberDF2.columns]).show()**



**Transformations:**

**Step 1 :Transformation**

**Transform logic: Prescriber Report:**

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

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

from pyspark.sql import functions as F

PrescriberDF2 = PrescriberDF2.filter(F.col("yearrs\_of\_exp").between(20,50))

PrescriberDF2.createOrReplaceTempView("prescriber")

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

A white background with black text

Description automatically generated

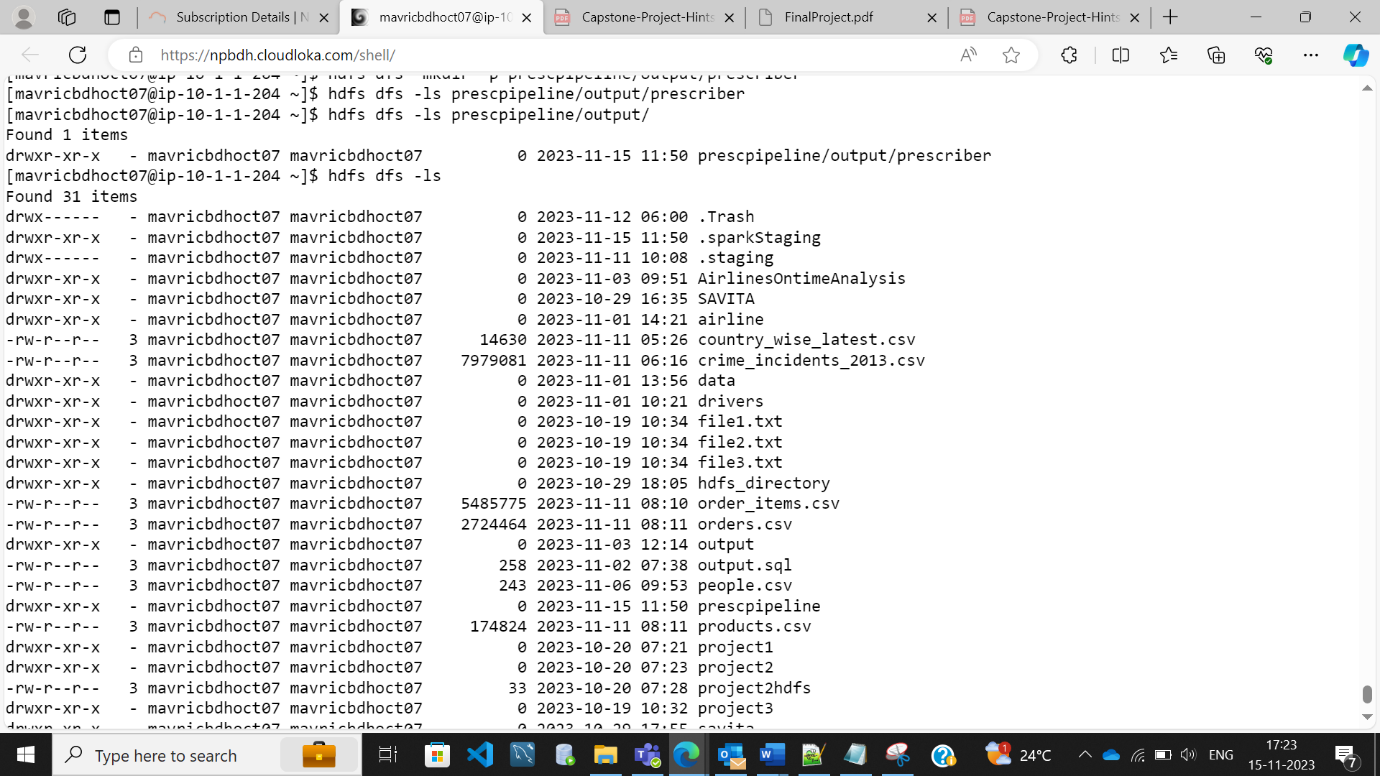
spark.sql("SELECT npi as Prescriber\_ID , np\_Fname as Prescriber\_Full\_Name, np\_state as Prescriber\_State, tot\_clm\_count as Total\_claim\_count, tot\_day\_supp as Total\_Days\_Supply, tot\_drug\_cst as Total\_Drug\_Cost from prescriber where tot\_claim\_cnt\_rank<= 5").show()

A screenshot of a computer

Description automatically generated

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

hdfs dfs -mkdir -p prescpipeline/output/prescriber



A screenshot of a computer

Description automatically generated

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

from pyspark.sql import SparkSession

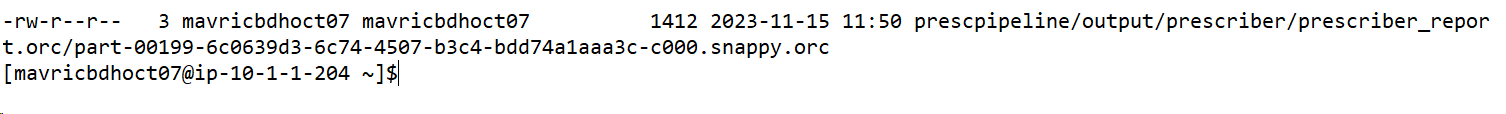
spark=SparkSession.builder.appName("SparkHive").enableHiveSupport().config("spark.sql.warehouse.dir","/user/hive/warehouse").getOrCreate()

spark.sparkContext.setLogLevel("ERROR")

prescriber\_report\_DF.write.mode("overwrite").option("compression","snappy").orc("/user/mavricbdhoct07/prescpipeline/output/prescriber/prescriber\_report.orc")

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

hdfs dfs -ls /user/mavericbdhoct07/prescpipeline/output/prescriber/prescriber\_report.orc/



prescriber\_report\_DF.write.partitionBy("tot\_clm\_count").mode("overwrite").saveAsTable("savitadb. prescriber\_report\_DF ")

use shrutid\_db;;

show tables;

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

