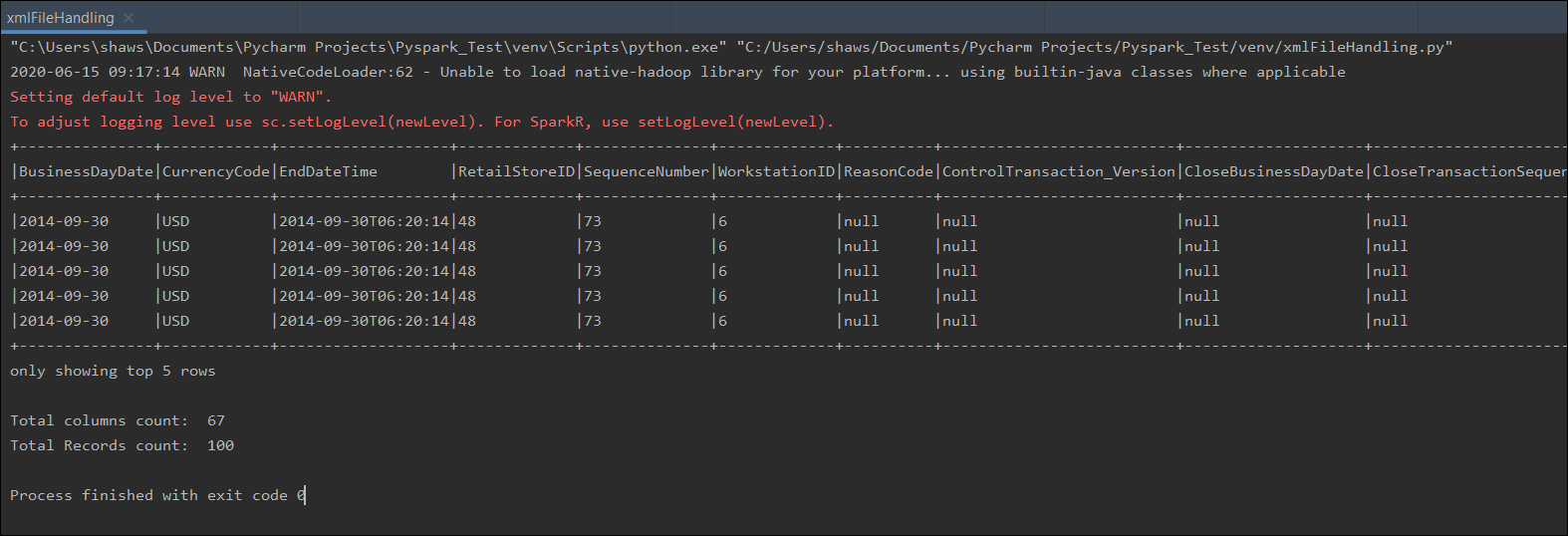
**Task 1: Flattening Transactions XML file**

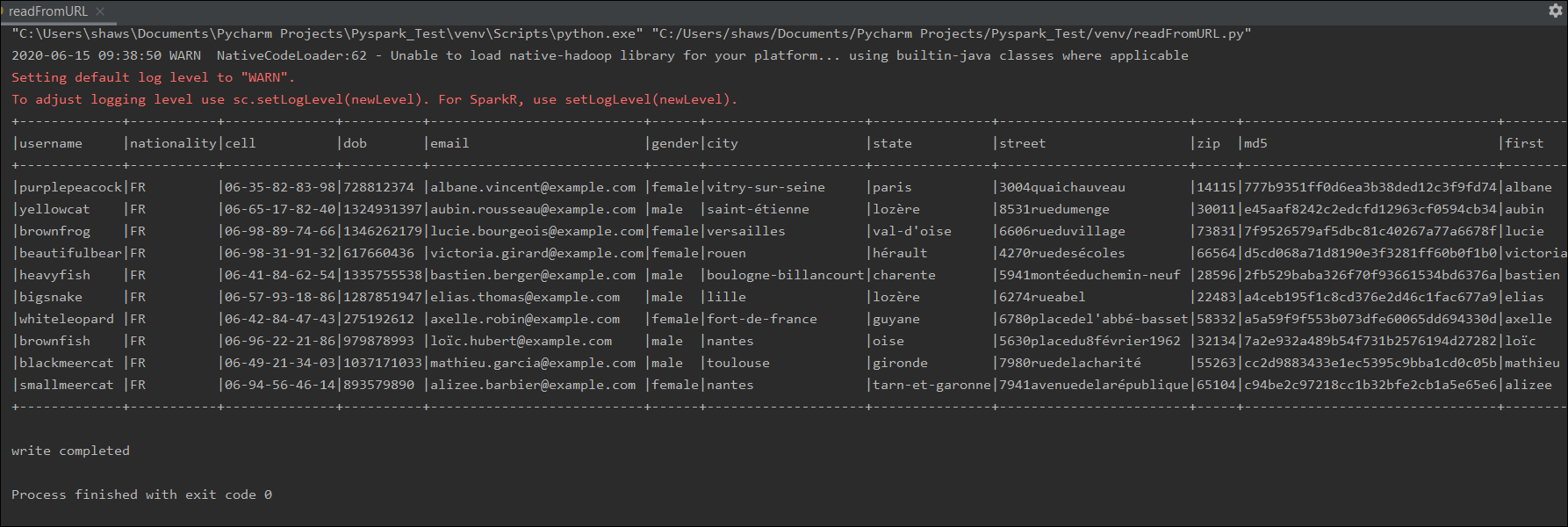
import os  
os.environ['PYSPARK\_SUBMIT\_ARGS'] = '--jars file:///C:/Users/shaws/Documents/Zeyobron/ExternalLibraries/\*.jar pyspark-shell'  
  
from pyspark import SparkContext,SparkConf  
from pyspark.sql import \*  
from pyspark.sql.types import \*  
from pyspark.sql.functions import \*  
import urllib.request  
  
def main():  
 sc = SparkContext(master="local", appName="test")  
 sc.setLogLevel("Error")  
 spark = SparkSession.builder.getOrCreate()  
  
 rawDF= spark.read.format("com.databricks.spark.xml")\  
 .option("rowTag","Transaction")\  
 .load("file:///C:/Users/shaws/Documents/Zeyobron/DataFiles/transactions.xml")  
   
 #-----------------------------------Flattening of XML Data-----------------------------------  
 cleansedDF= rawDF.selectExpr("\*","ControlTransaction.\*")\  
 .withColumnRenamed("\_Version","ControlTransaction\_Version")\  
 .selectExpr("\*","OperatorSignOff.\*")\  
 .selectExpr("\*","OperatorID.\*")\  
 .withColumnRenamed("\_OperatorName","OperatorID\_OperatorName") \  
 .withColumnRenamed("\_VALUE", "OperatorID\_VALUE")\  
 .selectExpr("\*","RetailTransaction.\*")\  
 .withColumnRenamed("\_Version","RetailTransaction\_Version")\  
 .withColumn("LineItem",explode(col("LineItem")))\  
 .selectExpr("\*","LineItem.\*")\  
 .withColumnRenamed("\_EntryMethod","LineItem\_EntryMethod") \  
 .withColumnRenamed("\_weightItem", "LineItem\_weightItem") \  
 .selectExpr("\*","PerformanceMetrics.\*")\  
 .withColumn("Total",explode(col("Total")))\  
 .selectExpr("\*","Total.\*")\  
 .withColumnRenamed("\_TotalType","Total\_TotalType")\  
 .withColumnRenamed("\_VALUE","Total\_VALUE")\  
 .selectExpr("\*","Sale.\*")\  
 .withColumnRenamed("\_ItemType","Sale\_ItemType")\  
 .selectExpr("\*","Tax.\*")\  
 .withColumnRenamed("\_TaxDescription","Tax\_TaxDescription")\  
 .withColumnRenamed("\_TaxID","Tax\_TaxID")\  
 .selectExpr("\*","Tender.\*")\  
 .withColumnRenamed("\_TenderDescription","Tender\_TenderDescription")\  
 .withColumnRenamed("\_TenderType","Tender\_TenderType")\  
 .withColumnRenamed("\_TypeCode","Tender\_TypeCode")\  
 .selectExpr("\*","Itemizers.\*")\  
 .withColumnRenamed("\_FoodStampable","Itemizers\_FoodStampable")\  
 .withColumnRenamed("\_Itemizer6","Itemizers\_Itemizer6")\  
 .withColumnRenamed("\_Itemizer8","Itemizers\_Itemizer8")\  
 .withColumnRenamed("\_Tax1","Itemizers\_Tax1")\  
 .withColumnRenamed("\_VALUE","Itemizers\_VALUE")\  
 .selectExpr("\*","MerchandiseHierarchy.\*")\  
 .withColumnRenamed("\_DepartmentDescription","MerchandiseHierarchy\_DepartmentDescription")\  
 .withColumnRenamed("\_Level","MerchandiseHierarchy\_Level")\  
 .withColumnRenamed("\_VALUE","MerchandiseHierarchy\_VALUE")\  
 .selectExpr("\*","POSIdentity.\*") \  
 .withColumnRenamed("\_POSIDType","POSIdentity\_POSIDType") \  
 .selectExpr("\*","Authorization.\*") \  
 .withColumnRenamed("\_ElectronicSignature","Authorization\_ElectronicSignature") \  
 .withColumnRenamed("\_HostAuthorized","Authorization\_HostAuthorized") \  
 .drop(col("ControlTransaction"))\  
 .drop(col("OperatorSignOff")) \  
 .drop(col("OperatorID"))\  
 .drop(col("RetailTransaction"))\  
 .drop(col("LineItem"))\  
 .drop(col("PerformanceMetrics"))\  
 .drop(col("Total"))\  
 .drop(col("Sale"))\  
 .drop(col("Tax"))\  
 .drop(col("Tender"))\  
 .drop(col("Itemizers"))\  
 .drop(col("MerchandiseHierarchy"))\  
 .drop(col("POSIdentity"))\  
 .drop(col("Authorization"))  
  
 #cleansedDF.printSchema()  
 cleansedDF.show(5,False)  
  
 record\_c = cleansedDF.selectExpr("COUNT(1) AS COUNT").collect()[0]["COUNT"]  
  
 print("Total columns count: ",len(cleansedDF.columns))  
 print("Total Records count: ", record\_c)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()



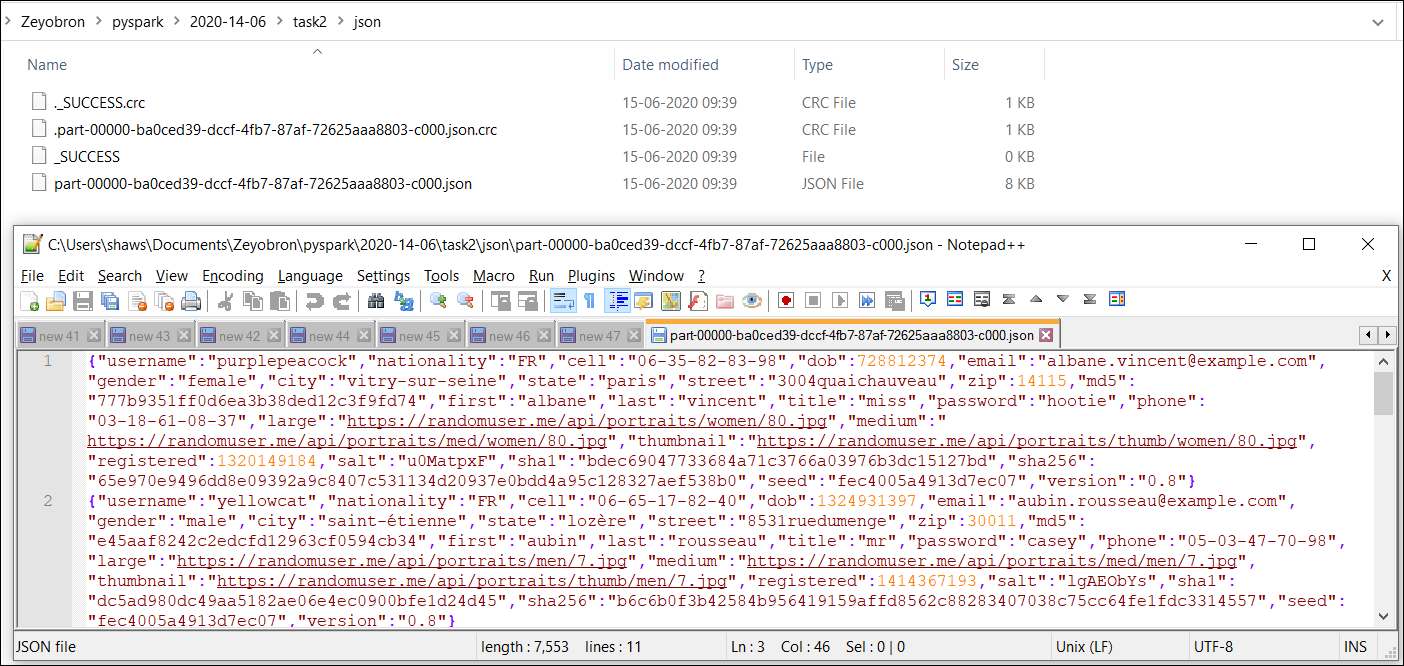
**Task 2: Read data from an URL and store in AVRO $ JSON format**

import os  
os.environ['PYSPARK\_SUBMIT\_ARGS'] = '--jars file:///C:/Users/shaws/Documents/Zeyobron/ExternalLibraries/\*.jar pyspark-shell'  
  
from pyspark import SparkContext,SparkConf  
from pyspark.sql import \*  
from pyspark.sql.types import \*  
from pyspark.sql.functions import \*  
import urllib.request  
  
def main():  
 sc = SparkContext(master="local", appName="test")  
 sc.setLogLevel("Error")  
 spark = SparkSession.builder.getOrCreate()  
  
 # -----------------------------Read data from URL-----------------------------  
 url = 'https://randomuser.me/api/0.8/?results=10'  
 response = urllib.request.urlopen(url)\  
 .read().decode('utf-8')  
 json\_str = str(response)  
  
 #-----------------------------String to Spark RDD-----------------------------  
 json\_list = ''.join(json\_str.split()).split()  
 rawRDD = sc.parallelize(json\_list)  
  
 # -------------------------Spark RDD to Spark Dataframe-----------------------  
 json\_df = spark.read.json(rawRDD)  
 #json\_df.printSchema()  
  
 # ------------------------------Flattening JSON-------------------------------  
 df = json\_df \  
 .withColumn("results",explode(col("results"))) \  
 .selectExpr("nationality","results.\*","seed","version") \  
 .selectExpr("nationality","user.\*","seed","version") \  
 .selectExpr("username","nationality","cell","dob","email","gender","location.\*",  
 "md5","name.\*","password","phone","picture.\*","registered",  
 "salt","sha1","sha256","seed","version") \  
 .withColumn("username",regexp\_replace(col("username"),"[0-9]",""))  
  
 df.show(20,False)  
  
 # ------------------------------Save into JSON-------------------------------  
 df.coalesce(1).write.format("json").mode("overwrite").save("file:///C:/Users/shaws/Documents/Zeyobron/pyspark/2020-14-06/task2/json")  
  
 # ------------------------------Save into JSON-------------------------------  
 df.coalesce(1).write.format("com.databricks.spark.avro").mode("overwrite").save("file:///C:/Users/shaws/Documents/Zeyobron/pyspark/2020-14-06/task2/avro")  
  
 print("write completed")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

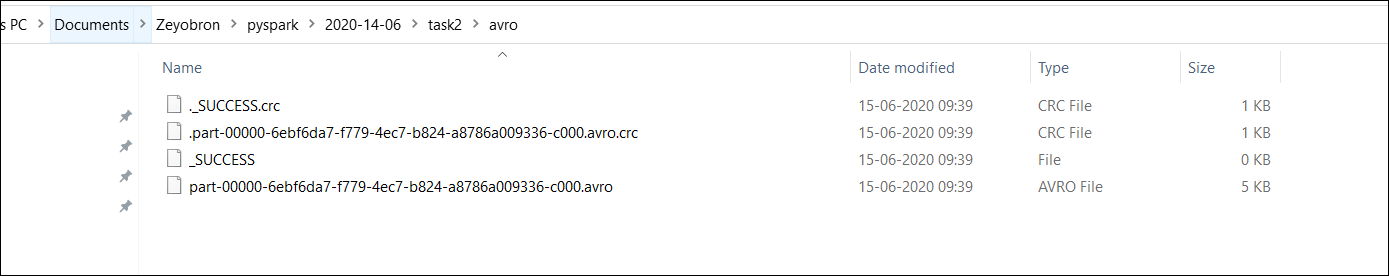
**Remove NUMERICALS from Column: username**



**JSON**

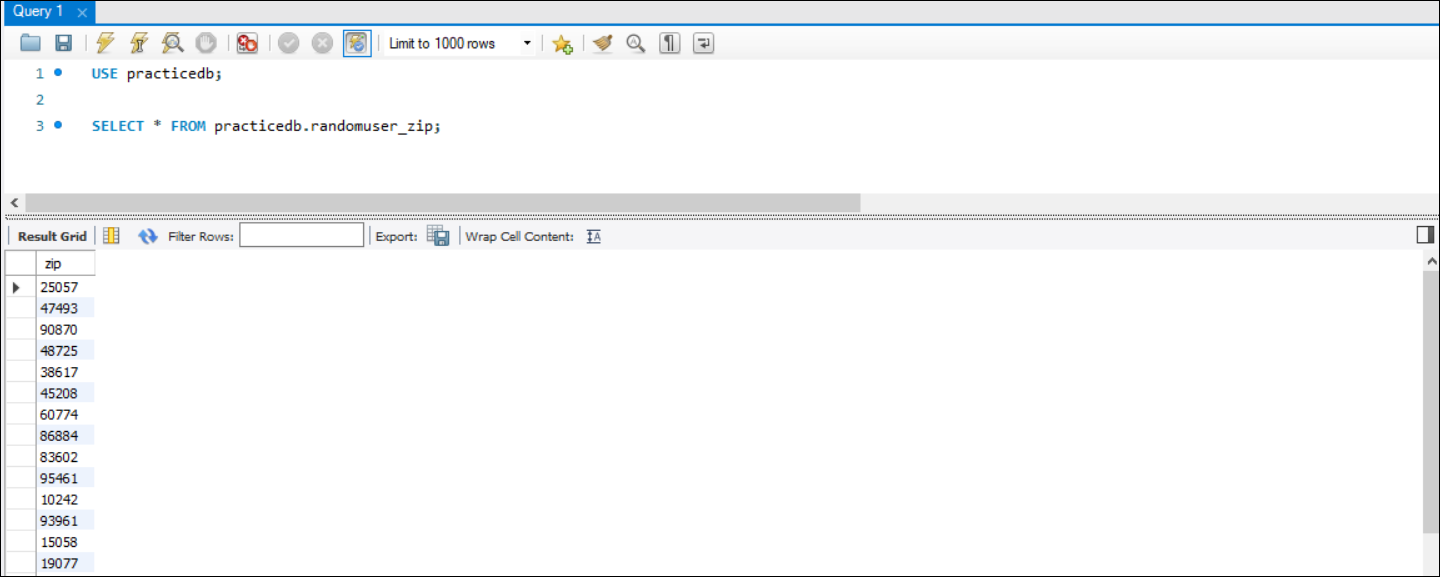


**AVRO**



**Task 3: Read data from URL and load ZIPCODE into MySQL table**

import os  
os.environ['PYSPARK\_SUBMIT\_ARGS'] = '--jars file:///C:/Users/shaws/Documents/Zeyobron/ExternalLibraries/mysql-connector-java-5.1.49.jar pyspark-shell'  
  
from pyspark import SparkContext,SparkConf  
from pyspark.sql import \*  
from pyspark.sql.types import \*  
from pyspark.sql.functions import \*  
import urllib.request  
  
def main():  
 sc = SparkContext(master="local", appName="test")  
 sc.setLogLevel("Error")  
 spark = SparkSession.builder.getOrCreate()  
 #.config("spark.driver.extraClassPath","C:/Users/shaws/Documents/Zeyobron/ExternalLibraries/mysql-connector-java-5.1.49.jar")\  
  
 # -----------------------------Read data from URL-----------------------------  
 url = 'https://randomuser.me/api/0.8/?results=40'  
 response = urllib.request.urlopen(url)\  
 .read().decode('utf-8')  
 json\_str = str(response)  
  
 #-----------------------------String to Spark RDD-----------------------------  
 json\_list = ''.join(json\_str.split()).split()  
 rawRDD = sc.parallelize(json\_list)  
  
 # -------------------------Spark RDD to Spark Dataframe-----------------------  
 json\_df = spark.read.json(rawRDD)  
 #json\_df.printSchema()  
  
 # ------------------------------Flattening JSON-------------------------------  
 df = json\_df \  
 .withColumn("results",explode(col("results"))) \  
 .selectExpr("nationality","results.\*","seed","version") \  
 .selectExpr("nationality","user.\*","seed","version") \  
 .selectExpr("location.zip") \  
  
  
 df.show(5,False)  
  
 # ------------------------------Check Table Count-------------------------------  
 count\_v = spark.read.format("jdbc").option("url","jdbc:mysql://localhost:3306/practicedb").option("driver", "com.mysql.jdbc.Driver")\  
 .option("user", "root").option("password", "admin").option("dbtable", "(SELECT COUNT(1) AS COUNT FROM practicedb.randomuser\_zip) t1\_alias").load()  
  
 x = count\_v.collect()[0]["COUNT"]  
 print("Total Record Count: ", x)  
   
 # ------------------------------Table Load-------------------------------  
 if (x >= 100):  
 df.write \  
 .format("jdbc") \  
 .option("url", "jdbc:mysql://localhost:3306/practicedb") \  
 .option("driver", "com.mysql.jdbc.Driver") \  
 .option("user", "root") \  
 .option("password", "admin") \  
 .mode("overwrite") \  
 .option("dbtable", "randomuser\_zip") \  
 .save()  
 else:  
 df.write \  
 .format("jdbc") \  
 .option("url", "jdbc:mysql://localhost:3306/practicedb") \  
 .option("driver", "com.mysql.jdbc.Driver") \  
 .option("user", "root") \  
 .option("password", "admin") \  
 .mode("append") \  
 .option("dbtable", "randomuser\_zip") \  
 .save()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()



**Task 4: Flattening a File**

from pyspark import SparkContext,SparkConf  
from pyspark.sql import \*  
from pyspark.sql.types import \*  
from pyspark.sql.functions import \*  
  
def main():  
 sc = SparkContext(master="local", appName="test")  
 sc.setLogLevel("Error")  
 spark = SparkSession.builder.getOrCreate()  
  
 rawDF = spark.read.format("csv").option("Delimiter","~").option("header","true").option("inferSchema","true")\  
 .load("file:///C:/Users/shaws/Documents/Zeyobron/DataFiles/PractitionerLatest.txt")  
  
 #--------------------------------------Split Column to Explode into Multiple Rows--------------------------------------  
 explodedDF= rawDF.withColumn("PROV\_PH\_DTL",split(col("PROV\_PH\_DTL"),"\\$").cast("array<string>"))\  
 .withColumn("PROV\_PH\_DTL",explode\_outer(col("PROV\_PH\_DTL")))\  
 .withColumn("PROV\_ADDR\_DTL", split(col("PROV\_ADDR\_DTL"), "\\$").cast("array<string>")) \  
 .withColumn("PROV\_ADDR\_DTL", explode\_outer(col("PROV\_ADDR\_DTL"))) \  
 .withColumn("PROV\_PH\_DTL", regexp\_replace(col("PROV\_PH\_DTL"), "^[ \\t]+|[ \\t]+$", ""))\  
 .withColumn("PROV\_ADDR\_DTL", regexp\_replace(col("PROV\_ADDR\_DTL"), "^[ \\t]+|[ \\t]+$", ""))  
  
 # --------------------------------------Split a Column to Convert into Multiple Columns--------------------------------------  
 cleansedDF= explodedDF.withColumn("PROV\_PH\_DTL",split(col("PROV\_PH\_DTL"),"\\|"))\  
 .withColumn("PROV\_PH\_DTL1",col("PROV\_PH\_DTL").getItem(0))\  
 .withColumn("PROV\_PH\_DTL2", col("PROV\_PH\_DTL").getItem(1))\  
 .withColumn("PROV\_PH\_DTL3", col("PROV\_PH\_DTL").getItem(2))\  
 .withColumn("PROV\_PH\_DTL4", col("PROV\_PH\_DTL").getItem(3))\  
 .drop(col("PROV\_PH\_DTL"))\  
 .withColumn("PROV\_ADDR\_DTL", split(col("PROV\_ADDR\_DTL"), "\\|")) \  
 .withColumn("PROV\_ADDR\_DTL1", col("PROV\_ADDR\_DTL").getItem(0)) \  
 .withColumn("PROV\_ADDR\_DTL2", col("PROV\_ADDR\_DTL").getItem(1)) \  
 .withColumn("PROV\_ADDR\_DTL3", col("PROV\_ADDR\_DTL").getItem(2)) \  
 .withColumn("PROV\_ADDR\_DTL4", col("PROV\_ADDR\_DTL").getItem(3)) \  
 .withColumn("PROV\_ADDR\_DTL5", col("PROV\_ADDR\_DTL").getItem(4)) \  
 .withColumn("PROV\_ADDR\_DTL6", col("PROV\_ADDR\_DTL").getItem(5)) \  
 .withColumn("PROV\_ADDR\_DTL7", col("PROV\_ADDR\_DTL").getItem(6)) \  
 .withColumn("PROV\_ADDR\_DTL8", col("PROV\_ADDR\_DTL").getItem(7)) \  
 .withColumn("PROV\_ADDR\_DTL9", col("PROV\_ADDR\_DTL").getItem(8)) \  
 .withColumn("PROV\_ADDR\_DTL10", col("PROV\_ADDR\_DTL").getItem(9)) \  
 .withColumn("PROV\_ADDR\_DTL11", col("PROV\_ADDR\_DTL").getItem(10)) \  
 .withColumn("PROV\_ADDR\_DTL12", col("PROV\_ADDR\_DTL").getItem(11)) \  
 .withColumn("PROV\_ADDR\_DTL13", col("PROV\_ADDR\_DTL").getItem(12)) \  
 .drop(col("PROV\_ADDR\_DTL"))  
  
 rawDF.filter(col("PROV\_ID")=="MAC000001896").select("PROV\_ID","PROV\_PH\_DTL","PROV\_ADDR\_DTL").show(20,False)  
 cleansedDF.filter(col("PROV\_ID")=="MAC000001896").select("PROV\_ID",  
 "PROV\_PH\_DTL1","PROV\_PH\_DTL2","PROV\_PH\_DTL3","PROV\_PH\_DTL4",  
 "PROV\_ADDR\_DTL1","PROV\_ADDR\_DTL2","PROV\_ADDR\_DTL3","PROV\_ADDR\_DTL4",  
 "PROV\_ADDR\_DTL5","PROV\_ADDR\_DTL6","PROV\_ADDR\_DTL7","PROV\_ADDR\_DTL8",  
 "PROV\_ADDR\_DTL9","PROV\_ADDR\_DTL10","PROV\_ADDR\_DTL11","PROV\_ADDR\_DTL12",  
 "PROV\_ADDR\_DTL13")\  
 .show(20,False)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

