Data Ingestion:

City Report:

A screenshot of a computer

Description automatically generated

A black and white text

Description automatically generated

A white background with black text

Description automatically generated

Prescriber Report:

A screen shot of a computer program

Description automatically generated

A close up of a document

Description automatically generated

A screenshot of a computer program

Description automatically generated

Data Cleansing/Preprocessing

**City Report**

Step1:Selecting the required columns

A screenshot of a computer

Description automatically generated

Step2:Converting the required fields to upper case

A white background with black text

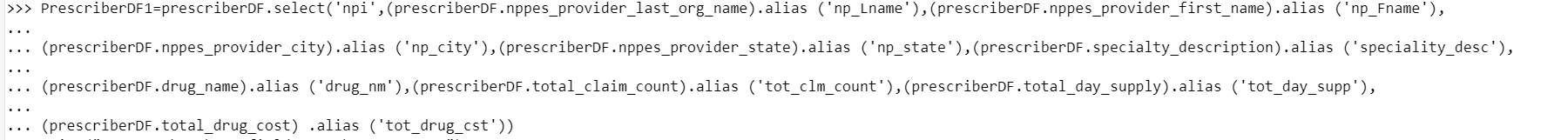
Description automatically generated

**Prescriber Report**

Step1:Selecting the required columns



Step2:Rename the above fields to shorter names



Step3:Add a Country Field 'USA' to the above data

A close-up of a letter

Description automatically generated

Step4:Clean the “years\_of\_exp” to extract only the numbers.

A close-up of a document

Description automatically generated

Removing special Characters from years of exp column:

A close-up of a document

Description automatically generated

Step6: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

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

PrescriberDF1.show(3)

A close-up of a document

Description automatically generated

Removing individual columns

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

PrescriberDF1.show(3)

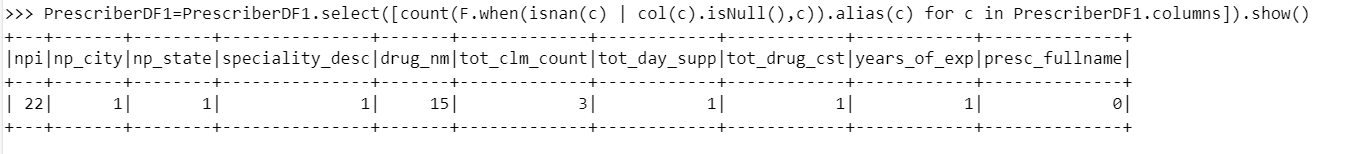
A close-up of a receipt

Description automatically generated

Step7: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()

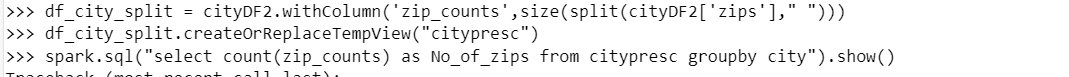


Data Transformation

**City Report**

Question 1

Calculate the Number of zips in each city.   
  
from pyspark.sql.functions import size  
df\_city\_split = city\_df\_sel.withColumn('zip\_counts',size(split(city\_df\_sel['zips']," ")))  
  
df\_city\_split.createOrReplaceTempView("citypresc")                                                                                            
spark.sql("select count(zip\_counts) as No\_of\_zips from citypresc groupby city").show()



A screenshot of a computer

Description automatically generated

Question 2

Calculate the number of distinct Prescribers assigned for each City.

PrescriberDF1.createOrReplaceTempView("presc")

spark.sql("select np\_city,np\_state,count(distinct(npi)),sum(tot\_clm\_count) from presc group by np\_city,np\_state").show()

A screenshot of a computer

Description automatically generated

Question -3

Calculate total TRX\_CNT prescribed for each city.  
spark.sql("select  np\_city,sum(tot\_clm\_count) from presc group by np\_city").show()

A screenshot of a computer

Description automatically generated

Question 4 - Do not report a city in the final report if no prescriber is assigned to it.

Spark.sql(“select city, count(npi) from df\_city join presc on df\_city.city=Presc.city group by city havning count(npi)>0”).show()

**Prescriber Report**

Consider the prescribers only from 20 to 50 years of experience

PrescriberDF3 = PrescriberDF2.groupBy(PrescriberDF2.presc\_id, PrescriberDF2.presc\_fullname, PrescriberDF2.presc\_state, PrescriberDF2.years\_of\_exp).agg(sum("total\_day\_supply").alias("total\_day\_supply"),

sum("total\_drug\_cost").alias("total\_drug\_cost"), sum("trx\_cnt").alias("trx\_cnt"))

spec = Window.partitionBy("presc\_state").orderBy(col("trx\_cnt").desc())

df\_presc\_final = PrescriberDF3.select("presc\_id", "presc\_fullname", "presc\_state",

"years\_of\_exp", "trx\_cnt", "total\_day\_supply", "total\_drug\_cost")\

.filter((PrescriberDF2.years\_of\_exp >= 20) & (PrescriberDF2.years\_of\_exp <= 50)) \

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

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

.select("presc\_id", "presc\_fullname", "presc\_state", "years\_of\_exp", "trx\_cnt",

"total\_day\_supply", "total\_drug\_cost")

Top 5 Prescribers with highest trx\_cnt per each state.

spark.sql("WITH Rankedpres AS (SELECT npi, np\_state, tot\_clm\_count, ROW\_NUMBER() OVER (PARTITION BY np\_state ORDER BY tot\_clm\_count DESC) AS rnk FROM pres)SELECT npi, np\_state, tot\_clm\_count FROM RankedPres WHERE rnk <= 5").show()