## A

## PROJECT REPORT

## ON

## “UberMedia Data Analysis”

Submitted for the partial fulfillment of the requirement for the

Award of the degree of

BACHELOR OF TECHNOLOGY

OF

INFORMATION TECHNOLOGY

(2015-2019)



UNDER SUPERVISION OF: - SUBMITTED BY: -

Dr. Sunila Godara Rajesh Kumar (15014029)

Associate Professor B. Tech (IT 8th SEM)

Deptt. Of CSE

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**GURU JAMBHESWAR UNIVERSITY OF SCIENCE & TECHNOLOGY**

**HISAR-125001**

**ACKNOWLEDGE**

I am very thankful to **Mrs. Sunila Godara** for her guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents for their kind co-operation and encouragement which help me in completion of this project.

I would like to express my special gratitude and thanks to industry persons for giving me such attention and time.

My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

**DECLARATION**

I, **Rajesh Kumar**, hereby declare that the project on “**Uber Media Data Analysis**” submitted by me to the **Guru Jambheswar University of Science and Technology**. The work contained in the report is original and has completely done by me.

We have followed the Guidelines provided by the University.

The work has not submitted to any institution for any degree or diploma. Whatever material we used (data, text or analysis) from other sources, we have given due credit to them in the text of the report and giving their detail in the references.

**RAJESH KUMAR 15014029**

**CERTIFICATE**

This is certified that **Rajesh Kumar (15014029)** has worked under my supervision to prepare the project on **“Uber Media Data Analysis”.** He has worked on this project through the semester July 2018 to April 2019.I wish him success in life.

**Dr. Sunila Godara**

Associate Professor

CSE Dept.

GJUS&T, Hisar

**Abstract**

The main idea of the project to build ETL pipelines, after that we are doing analysis of that data as like aggregation and join report.

In UberMedia project Data we got normally from these locations:

* Places visited/ Points of interest/ Common evening locations captured via mobile device signals. For Example- visit to an auto dealership, a home improvement retailer, jewelry shops, Malls etc.

This project we are done in two part: -

1)Raw Layer

2)Business Layer

Analysis on the first Row Layer and after that we are applying analysis on business layer.

We are making the jars using tools are sbt and maven.

After making jars using in the spark submit and give the path of the location which location of data are process and after processing which location data are sending.

we are using AWS S3 for storage purpose and making folder for which data are good records and data are bad records all files are storage in s3 bucket.

After this data are processed and sending the good records for future purpose.

After that we are submitted that data for future analysis.

**Content**

ACKNOWLEDGE

DECLARATION

PLAGIARISM CERTIFICATE

CERTIFICATE

ABSTRACT

**CONTENT**

1. INTRODUCTION 1
   1. INTRODUCTION 1
   2. PROJECT DESCRIPTION 2
   3. MOTIVATION 4
   4. OBJECTIVE 7
2. PROJECT OVERVIEW 8
3. SYSTEM REQUIREMENT 9
4. SYSTEM DESIGN10
5. IMPLEMENTATION 12
   1. METHODOLOGY 12
   2. TECHNOLOGY USED 12
   3. DATASET 14
   4. RESULT 15
6. CONCLUSION AND FUTURE SCOPE 24

REFERENCES25

APPENDIX 26

PLAGIARISM REPORT 65

**Introduction**

Writing Spark Jobs for Data Consumption from S3 buckets and transform it as per the requirement and then ingesting it into HDFS raw zone.

Built an Enterprise Data Lake by doing various aggregations as per the business requirement from HDFS raw zone data to provide a single abstracted view of business data to our client’s business units for advanced analytics.

The objective of UberMedia project is get insight of customers based on their geolocation data or through their visit using their device id (Hashed format). Data which we got contains latitude, longitude, Mobile device id in hashed format, polygon id and UNIX time.

So, we get customer analysis like:

Total number of visit customers

Max/max visit time of customers.

In this project we are using the uber data that means any person which are booked an uber drive the details of that person are in the uber data cluster.

In this detail the polygon\_id, latitude and longitude are using and analyze for this location what actually this location and which category this is considering.

**Project Description (features)**

Data we got normally from these locations:

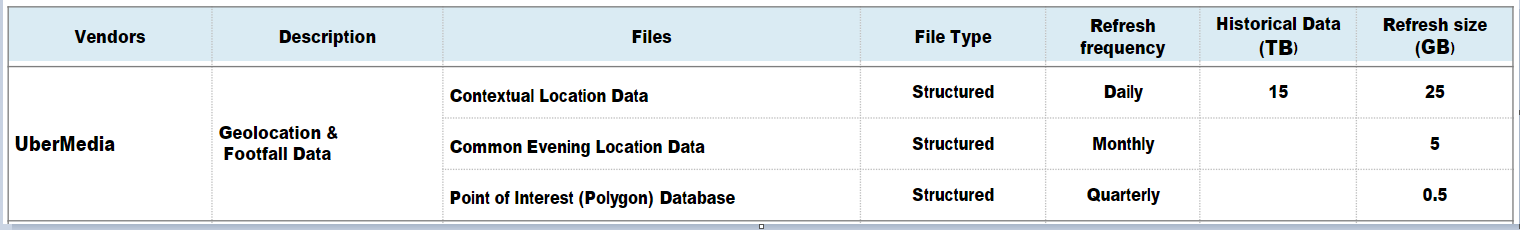
* Places visited/ Points of interest/ Common evening locations captured via mobile device signals. For Example- visit to an auto dealership, a home improvement retailer, jewelry shops, Malls etc.

Figure:-1(UberMedia Geolocation Data)

In ubermedia we got 3 types of data files

Contextual location data:

* CLD data is in structured format and occur daily.
* Which contains: device\_id, unixtime, latitude, longitude, polygon\_id.

Common Evening Location data:

* + Its same as CLD but its data of common location and occur monthly
  + Which contains: device\_id, country, last seen, latitude, longitude,
  + state, zip code, census.

Point of Interest data:

* It contains polygon data which occur quarterlyWhich contains: Polygon ID, Polygon Name, Full Address, Address1, Address2, City, State, Zip, Zip4, Polygon Category.
* These data we got in AWS S3 bucket and we read this data using spark then apply cleaning, transformation and then save PII information to S3 and Non PII information to HDFS/Hive.

For this process we require two layers: Raw layer and Business Layer

1)Raw Layer steps:

* Integration between S3 and spark.
* Reading data from S3 using spark sql
* Pre validation on data
* Header validation
* Custom filtration
* Data transformation and segregating good and bad records
* Write good records and Non PII to Hive/HDFS
* PII Records to S3
* Bad records in S3
* Post validation
* Audit
* Logs

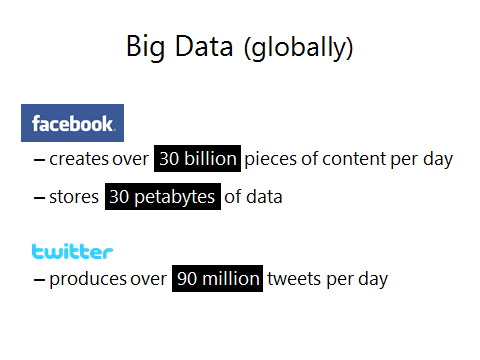
2) Business Layer steps involved:

* Integration between hive and spark.
* Pre validation on data
* data from hive with latest file date
* Perform Aggregation, timestamp transformation, data analysis.
* Post validation
* Audit

**Motivation**

**Big Data  
 What it is and why it matters: -**

* In today modern life are using big data is everywhere.
* Lots of data are being collected and warehoused from the system
* Web of data, electronic commerce
* Purchased by any authority or the grocery store
* Bank /credit card transaction system
* Social sites network system.



**IBM Definition**

Big data is created from everywhere every timein wholeworld. Every process and each social media process are creating it. Systems mobile devices are creatingthem

**What is Big Data?**

* + This data counted from everywhere: devices and sensors helpful incollectiveweather forecasting info, digital images and video formats, bought transaction data, and mobile phones trackingsignals to name some. That is called the big data.

**Competitive advantage **

Data is being generated from the new resources in the modern advantage

**Decision making**  

Decision making is move from the lite data flow.

**Value of data **

As the value of data continues grow in the newest industry .

**Vs of big data: -**

**The 5’vs of big data are**

**Volume: -** The size of the data.

**Velocity: -** The speed at the data generated.

**Variety: -** The data are various type & format (Structural/nonstructural).

**Veracity: -** The trust worthiness of the data in the terms of the accuracy.

**Value: -** We says just have big data is of no use whenever we are converted into value of that data.

**What to do with these data?**

Aggregation and Statistics

Data warehouse and OLAP

Indexing, Searching, and Querying

Keyword based search

Pattern matching (XML/RDF)

**Objective**

The main idea of the project to build ETL pipelines, after that we are doing analysis of that data as like aggregation and join report.

In UberMedia project Data we got normally from these locations:

* Places visited/ Points of interest/ Common evening locations captured via mobile device signals. For Example- visit to an auto dealership, a home improvement retailer, jewelry shops, Malls etc.

This project we are done in two part: -

1)Raw Layer

2)Business Layer

Analysis on the first Row Layer and after that we are applying analysis on business layer.

We are making the jars using tools are sbt and maven.

After making jars using in the spark submit and give the path of the location which location of data are process and after processing which location data are sending.

we are using AWS S3 for storage purpose and making folder for which data are good records and data are bad records all files are storage in s3 bucket.

After this data are processed and sending the good records for future purpose.

After that we are submitted that data for future analysis.

**Project Overview**

This project is based on the big data technology called spark with Scala. We are working this project on uber media data analysis. In this project additionally we are using cloud computing technology called amazon web services(aws).

This project we are completed in two parts means we are working this project on two layer one is raw layer and the other one is business layer.

Firstly, working on the raw layer in which we are using the raw data for analysis means uber data we are using to process and analysis this data.

Working on the ETL (Extract Transform & Load) in this complete project we are extract transform and load the data.

The using of uber data for future business purpose and in the modern-day life the data are using for banking, e-commerce and other sector like healthcare.

So, we are using various department of the industry analyzed data will be used for future growth of the industry.

**Software Requirements: -**

Tools/Software Required: -

* **Big data Technologies:** Hadoop, Spark, AWS (additional)
* **Hadoop Components:** HDFS, Hive
* **Cloud Components:** S3(additional)
* **Languages:**  Unix/Linux, Scala/Python
* **Database:** Hive
* **Build Tools:** SBT, Maven
* **IDE:** Eclipse, IntelliJ
* **Operating Systems:** MS Windows, Unix/Linux
* **Other tool:** MobaXterm, VMware, mapr

This project we are completed using some tools of big data.

This project we are completed in two ways -In the local server or the cloud server of aws(Amazon Web Server).We are using the local server for best performance of our project because in the cloud server some time are network error are occurring so that's why we are using the local VMware to execute of our project.

**System Design**

In the system design we are using spark submit for the run of the job and the create command are using for the table create which is created in hdfs database.

Spark Submit: -

spark-submit --class com.analytic.ubermedia.raw.ds\_um\_driver /home/mapr/jar/pipelines-0.0.1-SNAPSHOT.jar UM\_DAILY\_CONTEXTUAL\_LOCATION ubermedia/raw/cld/input/01JUN2019 /user/mapr/ubermedia/raw/cld/output\_non\_pii /user/mapr/ubermedia/raw/cld/error /user/mapr/ubermedia/raw/cld/archive /user/mapr/ubermedia/raw/cld/audit /user/mapr/ubermedia/raw/cld/output\_pii /user/mapr/ubermedia/raw/cld/logs

spark-submit --class com.analytic.ubermedia.raw.ds\_um\_driver /home/mapr/jar/pipelines-0.0.1-SNAPSHOT.jar UM\_POI /user/mapr/ubermedia/raw/poi/input/201906 uber.poi /user/mapr/ubermedia/raw/poi/error /user/mapr/ubermedia/raw/poi/archive /user/mapr/ubermedia/raw/poi/audit /user/mapr/ubermedia/raw/poi/pii\_table /user/mapr/ubermedia/raw/poi/logs

spark-submit --class com.analytic.um.bssn.ds\_um\_driver /home/mapr/jar/pipelines-0.0.1-SNAPSHOT.jar UM\_DAILY\_CONTEXTUAL\_LOCATION uber.poi uber.non\_pii uber.business /user/mapr/ubermedia/business/logs

**Table Create: -**

create external table non\_pii(

accenture\_device\_id string,

lat double,

lon double,

polygon\_id string,

unixtime string,

batch\_id string,

data\_collection\_date string,

input\_file\_name string,

event\_datetime string,

file\_date date

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

LOCATION '/user/mapr/ubermedia/raw/cld/output\_non\_pii';

create table poi

(

PolygonID string,PolygonName string,FullAddress string,Address1 string,Address2 string,City string,State string,Zip string,Zip4 string,PolygonCategory array<string>,input\_file\_name string,run\_date string,batch\_id string

)

partitioned by (file\_date string)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE;

create table uber.business

(

accenture\_device\_id string,lat\_lon string, mintime string,maxtime string,lat double,lon double,tzoneid string,mintime\_local string,maxtime\_local string,Polygon\_id string,PolygonCategory string,input\_file\_name string,batch\_id string,load\_date string

)

partitioned by (file\_date string,data\_collection\_date string)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE;

**Implementation**

**Methodology & Technology used: -**

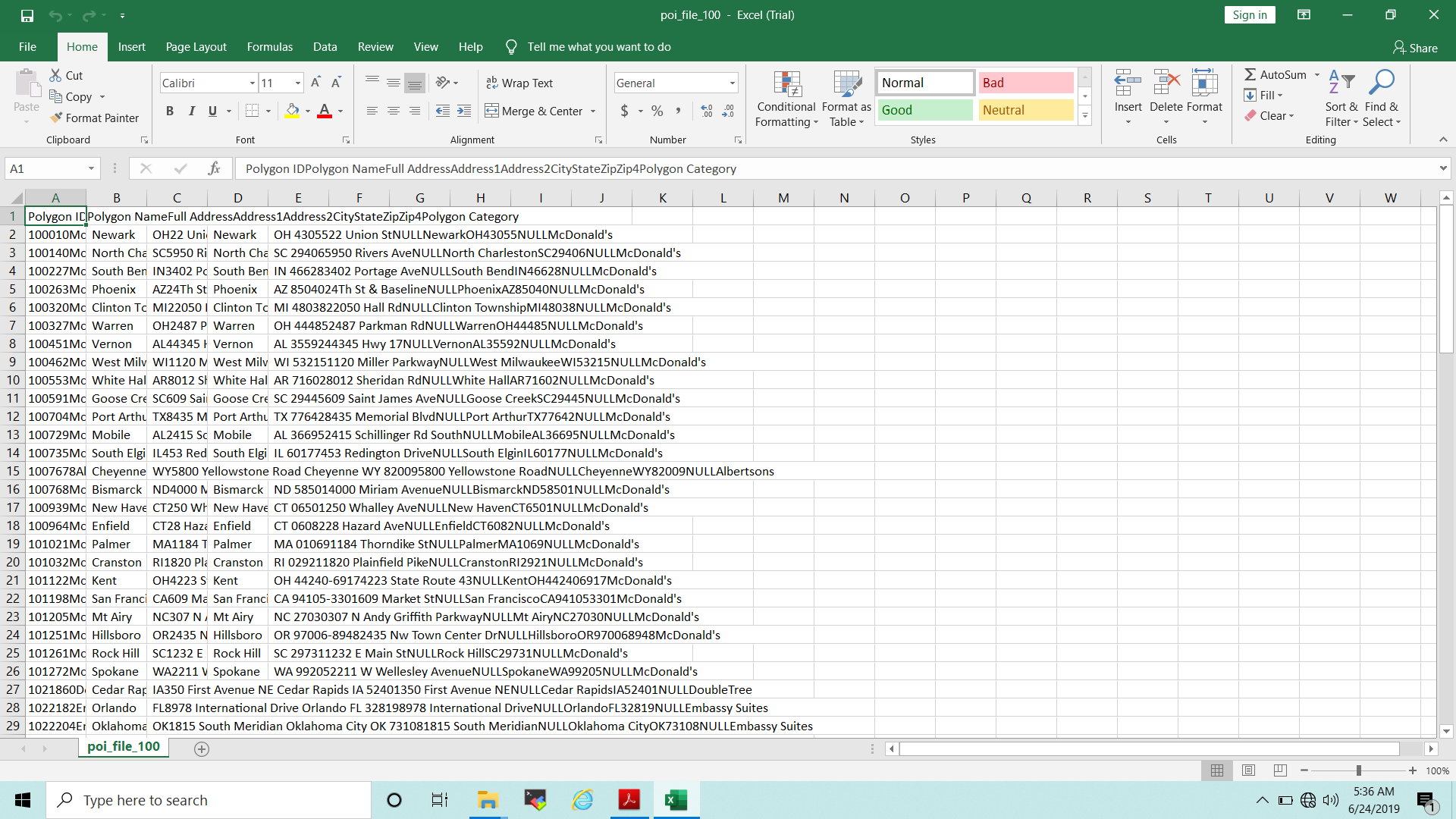
During the completion of this project we are using various method and technology are used. In this project we using the big data technologies and big data analysis methods.

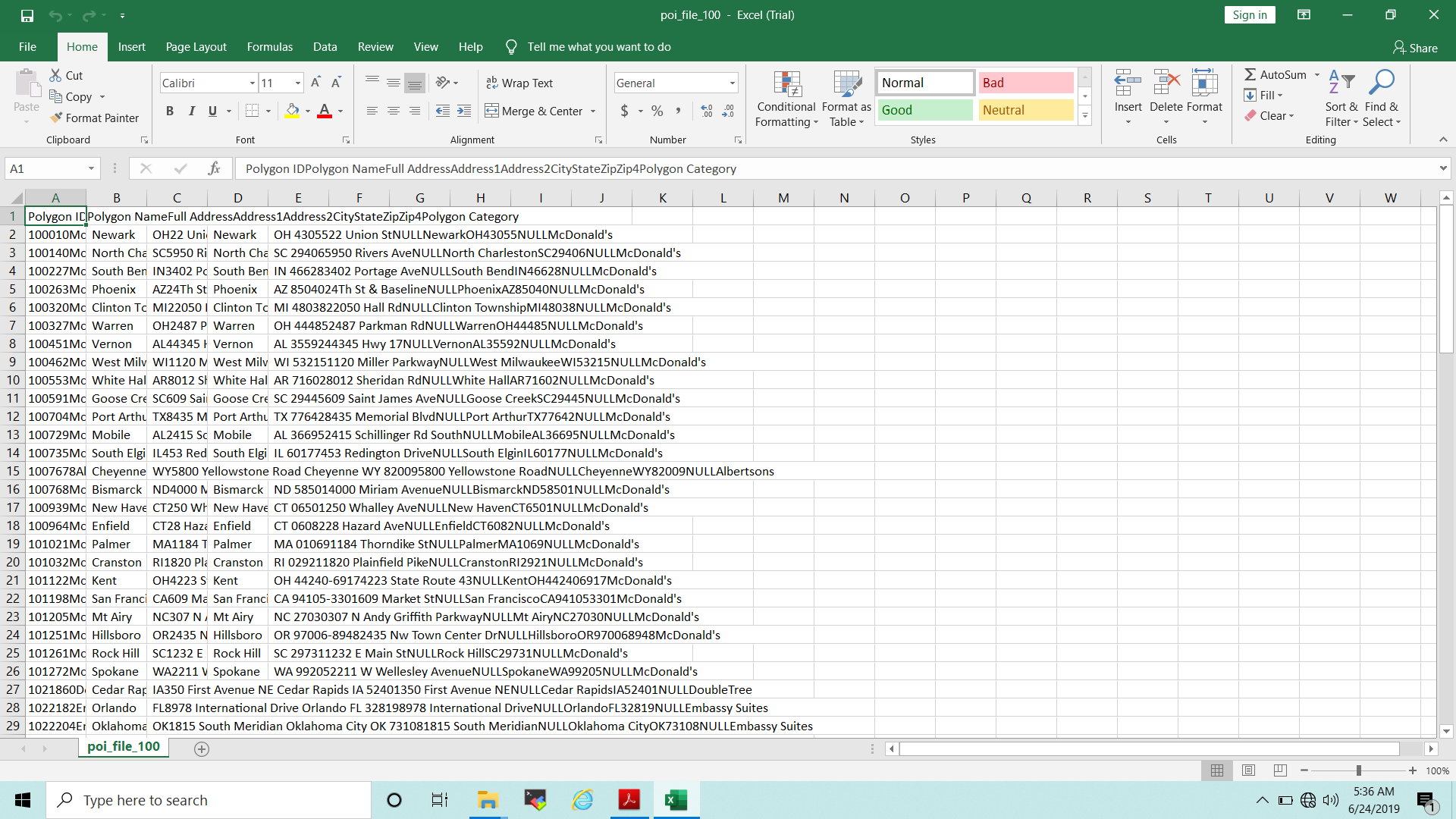
For the data ETL we are using the VMware for data processing and the using cloud tools mapr for the using of local services.

Inside the VMware we are using mapr after successfully login of mapr we are using the services in the local. And the one other software we are using of better data processing is the MobaXterm. Using of this tool we are easily process our data.

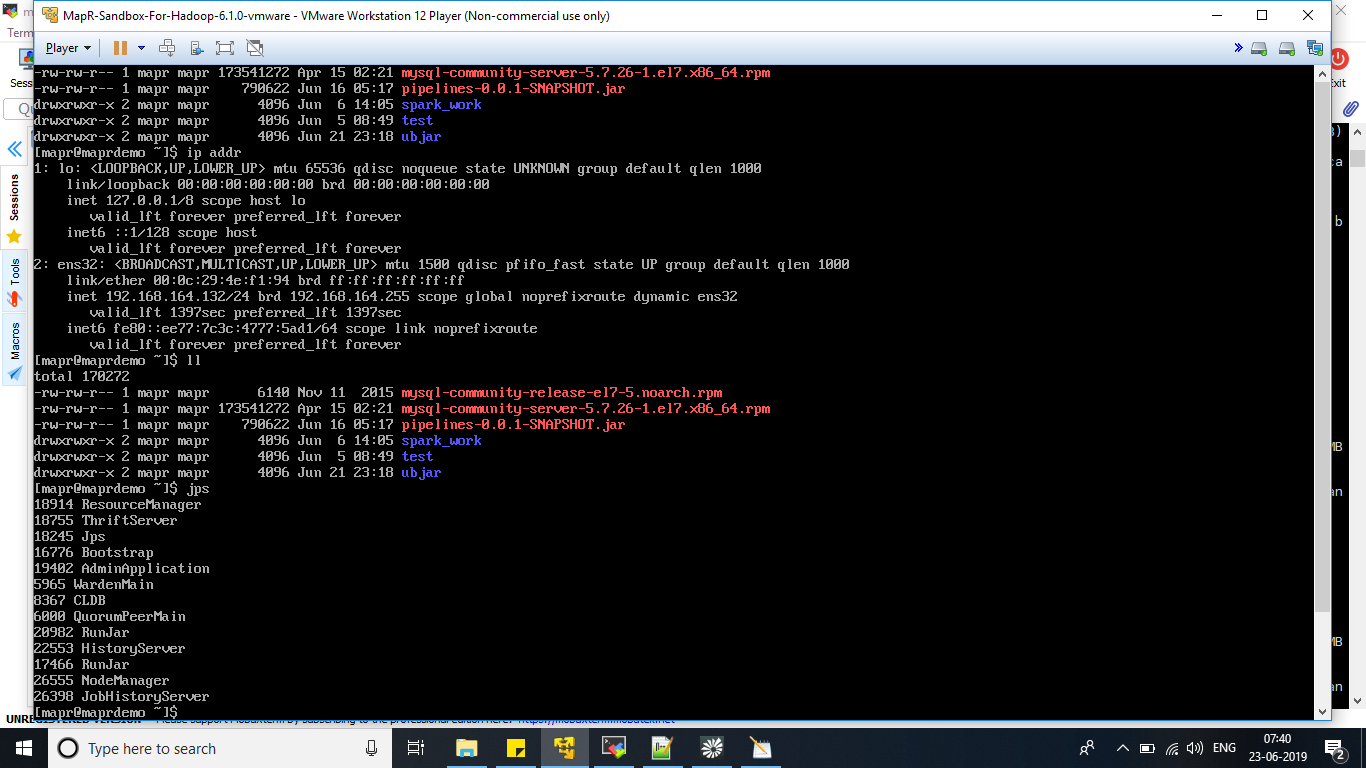
In the hdfs we are created directory for the contextual location data. In the hdfs we are created the directory for the raw layers and the business layers.

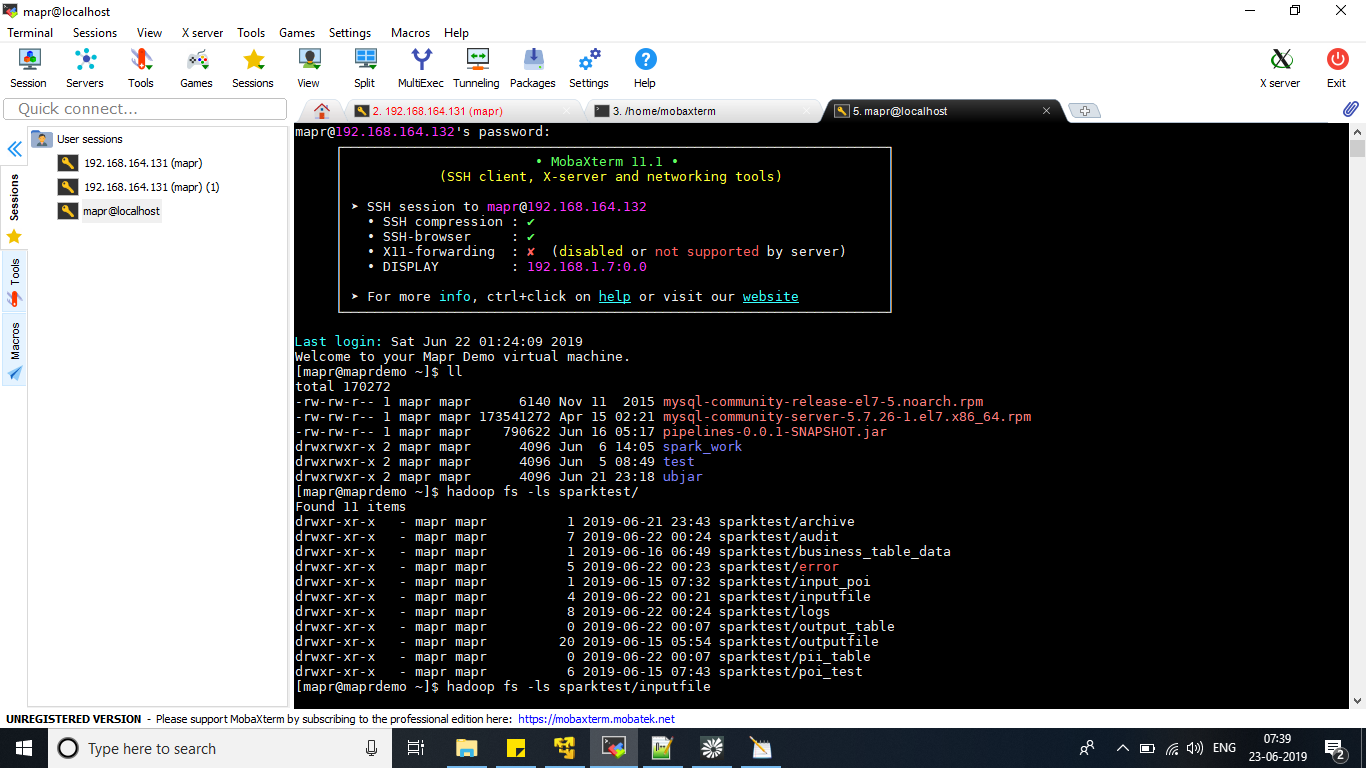
**Dataset: -**

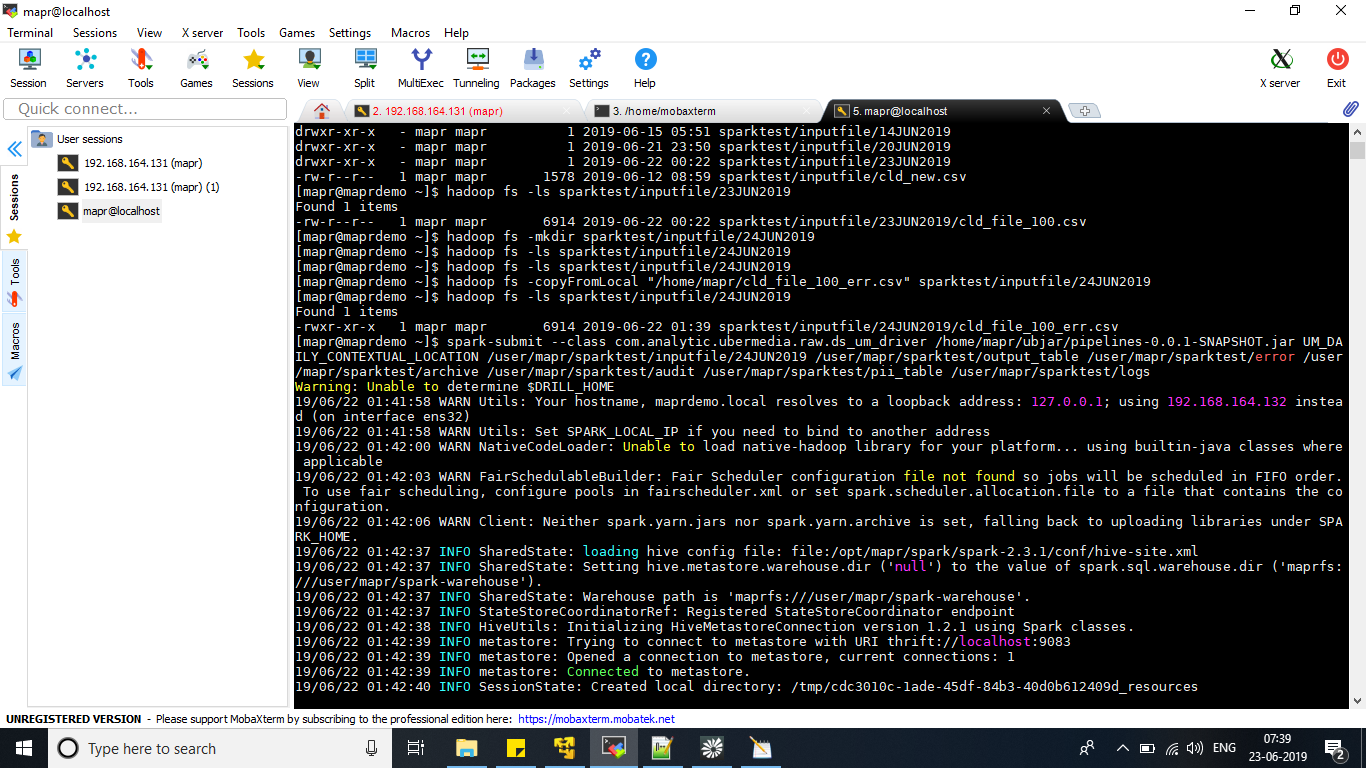


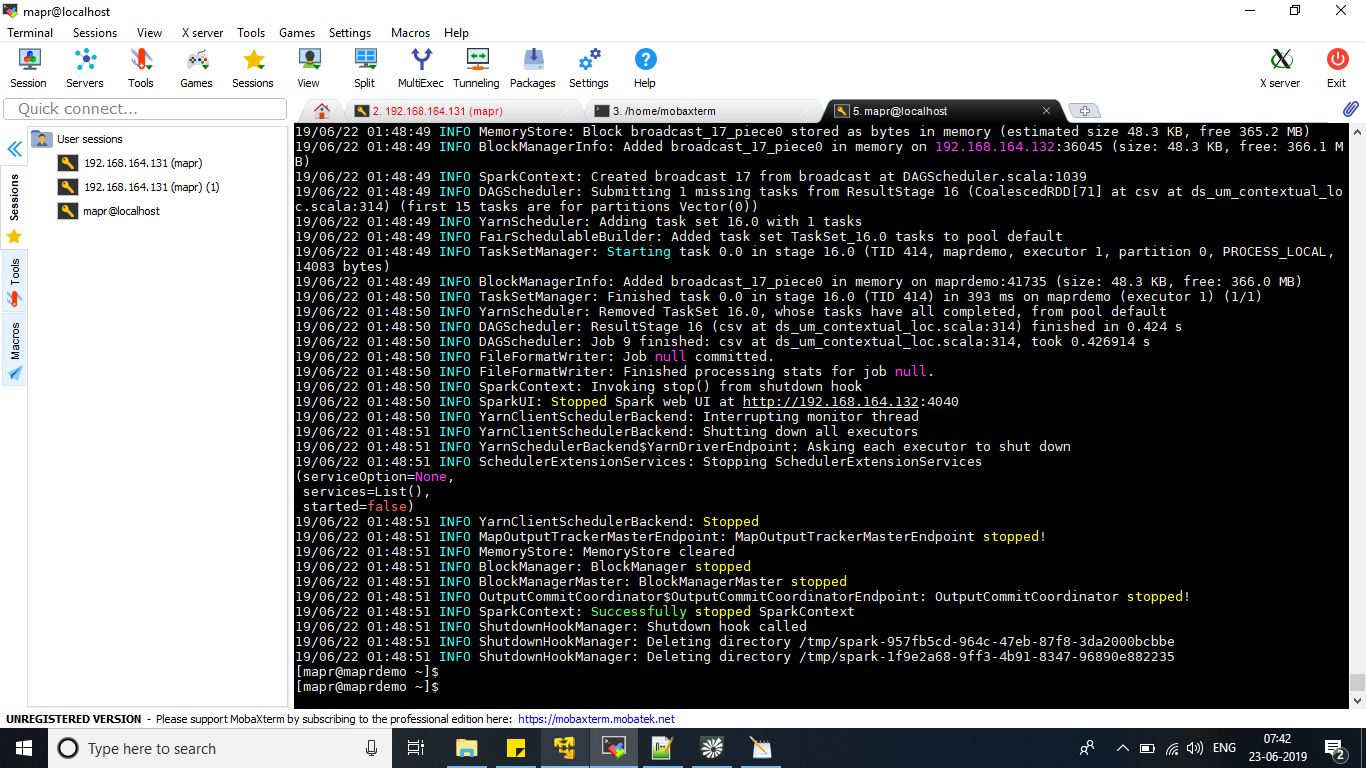


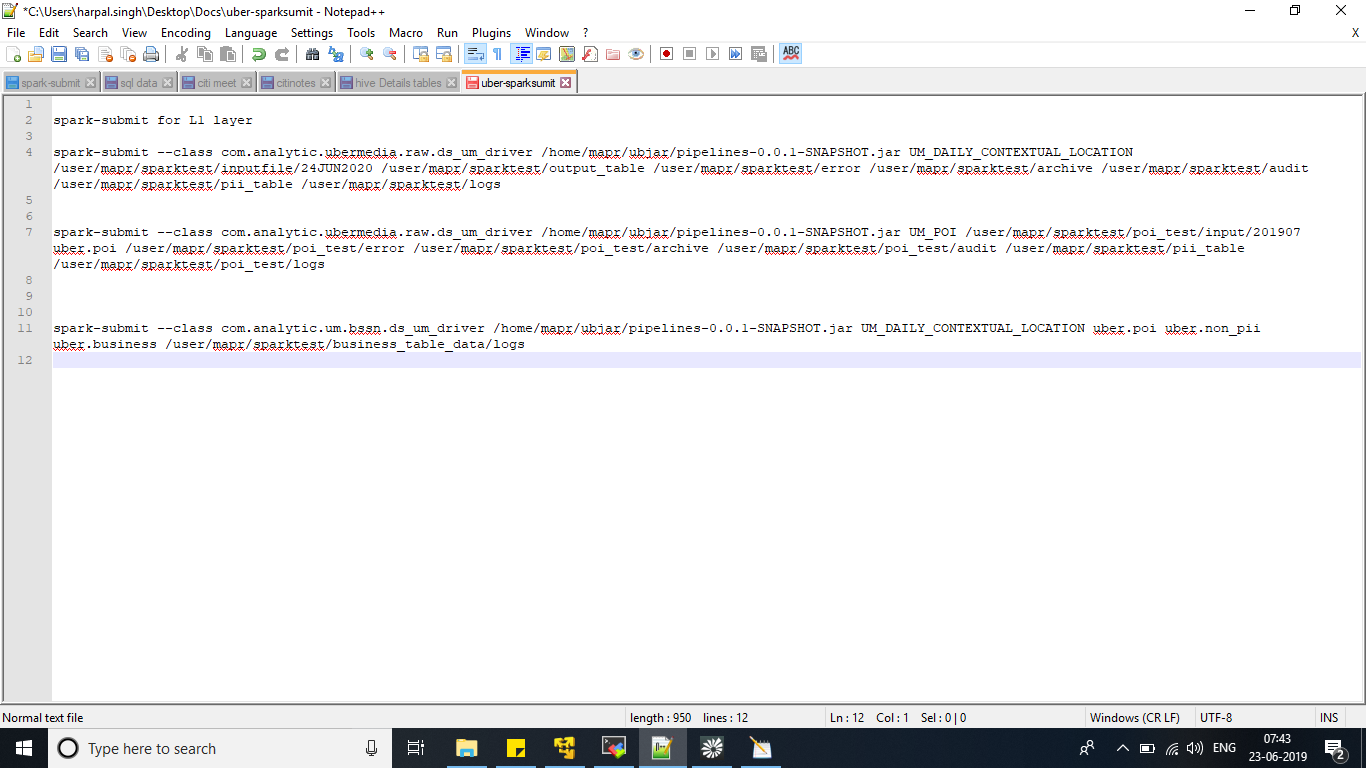
**Result**

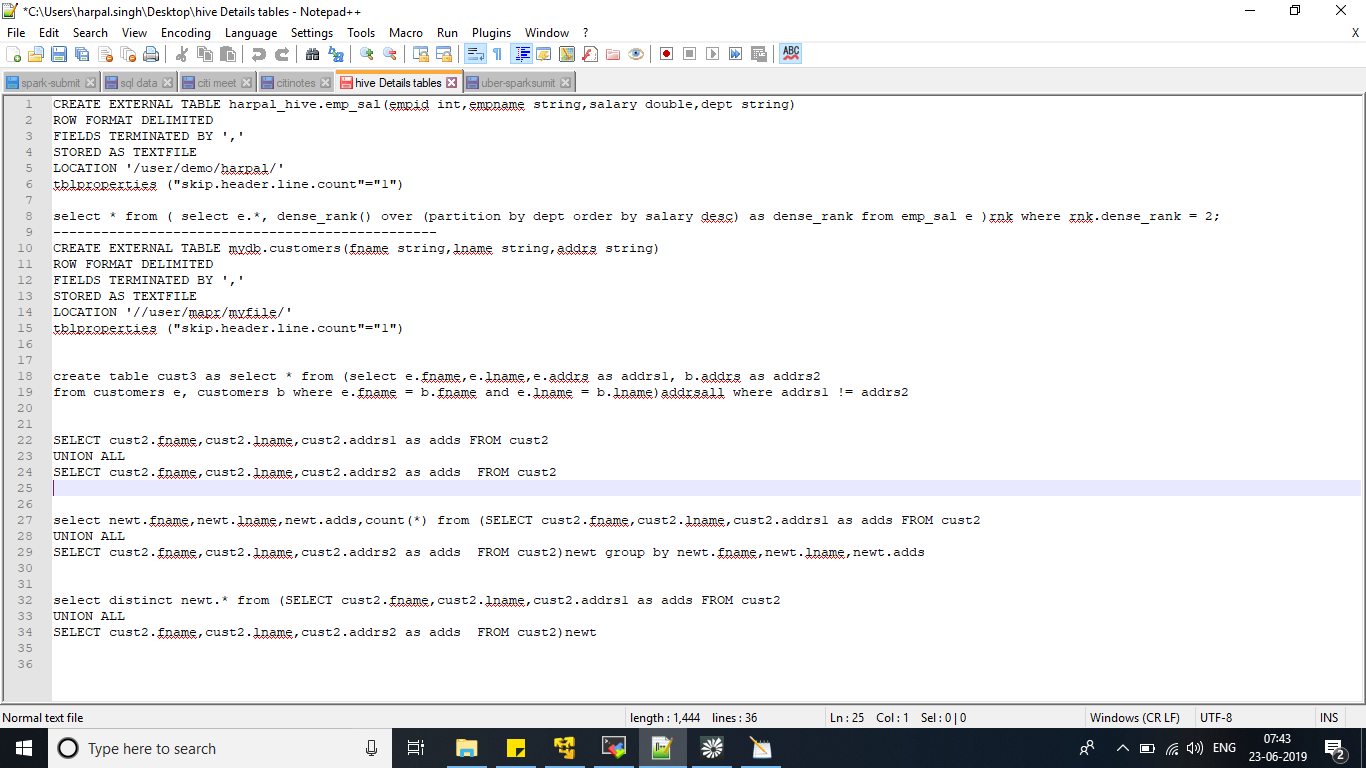


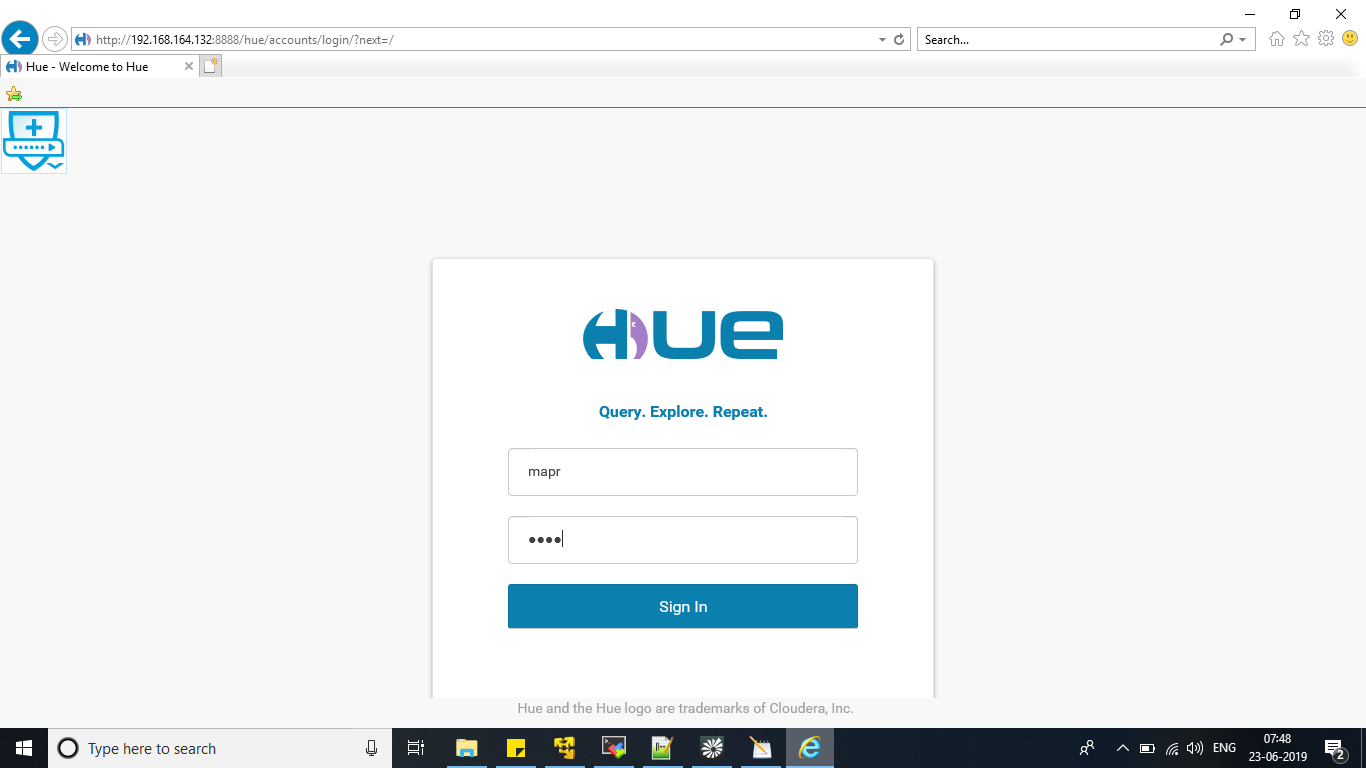


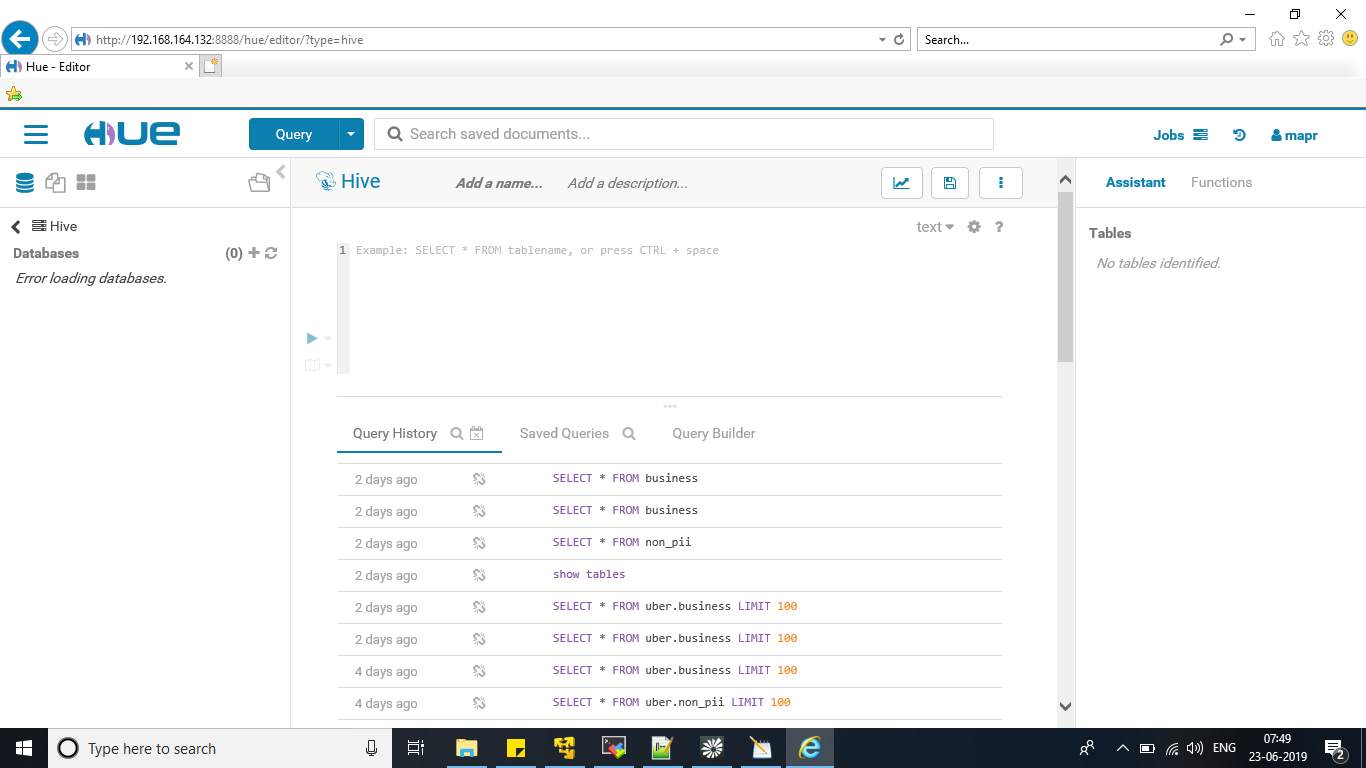


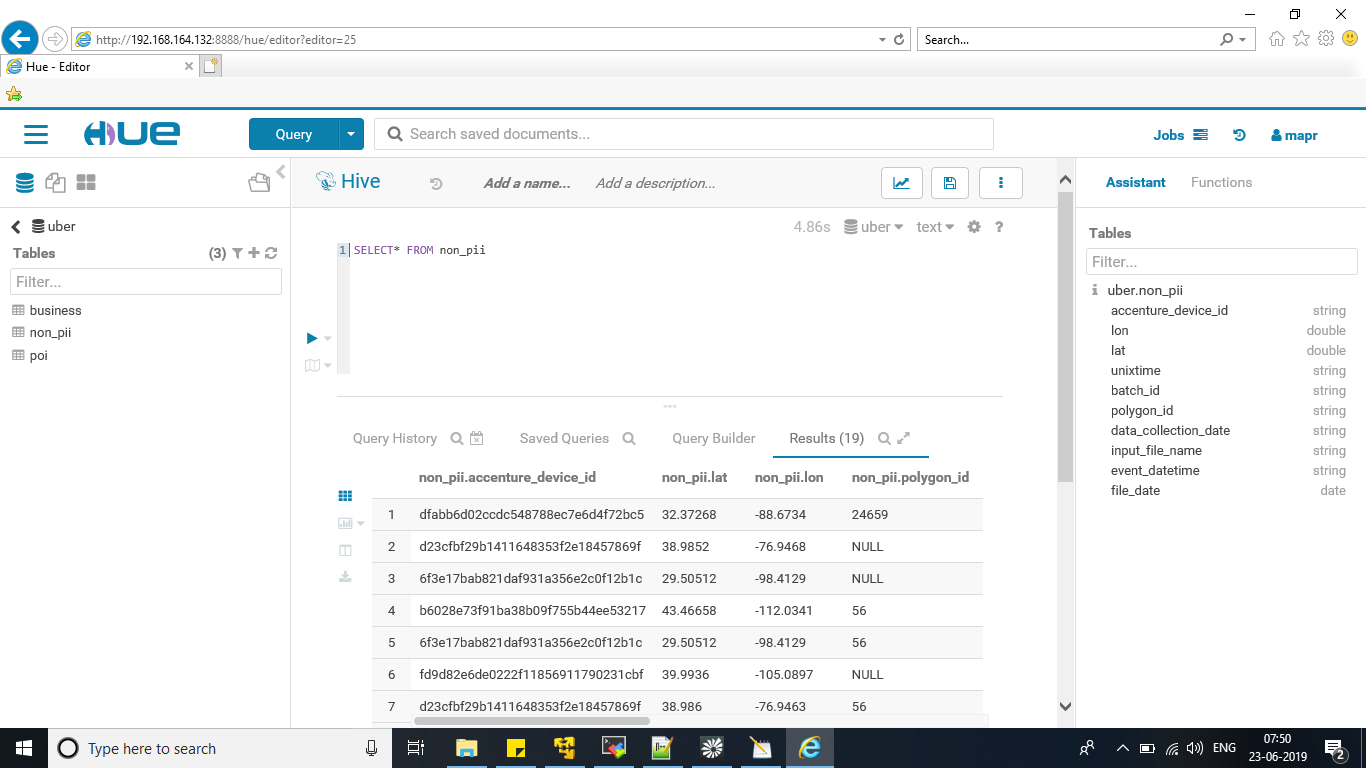


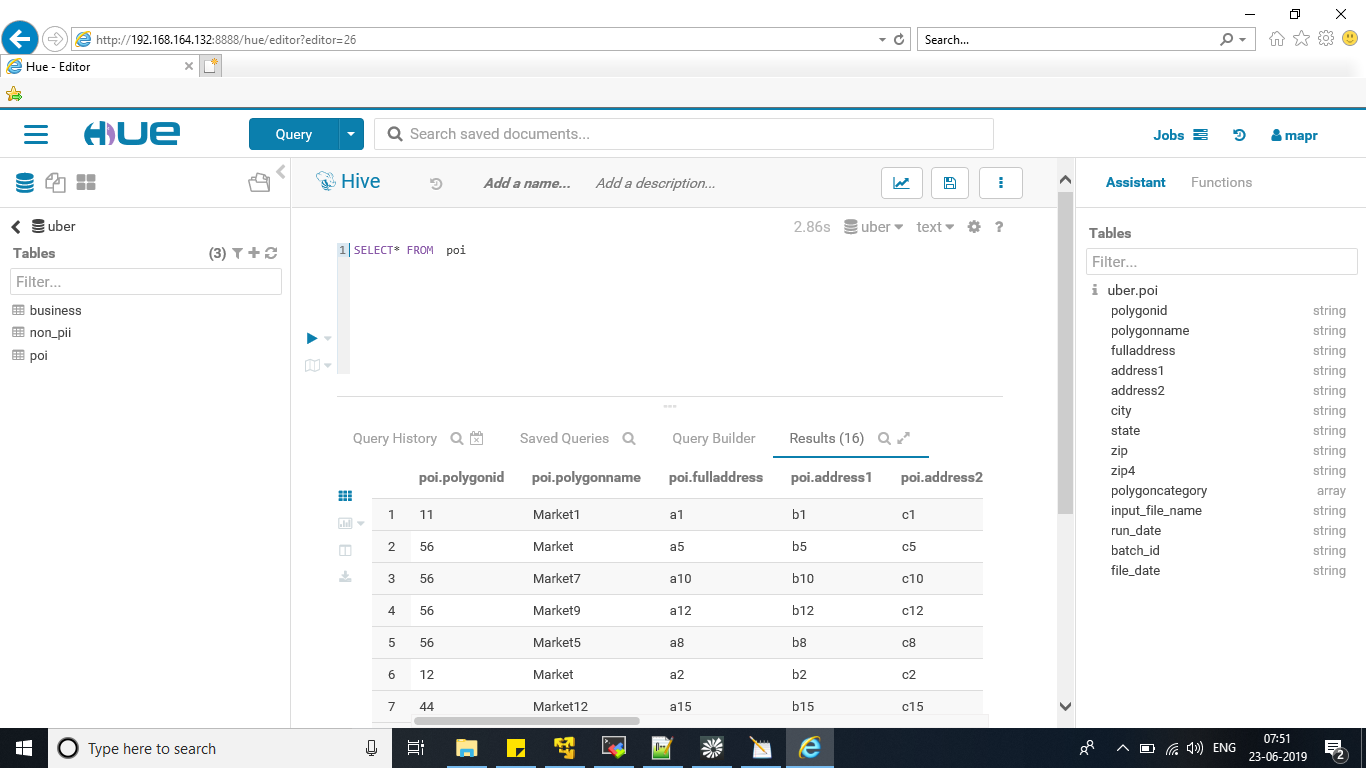


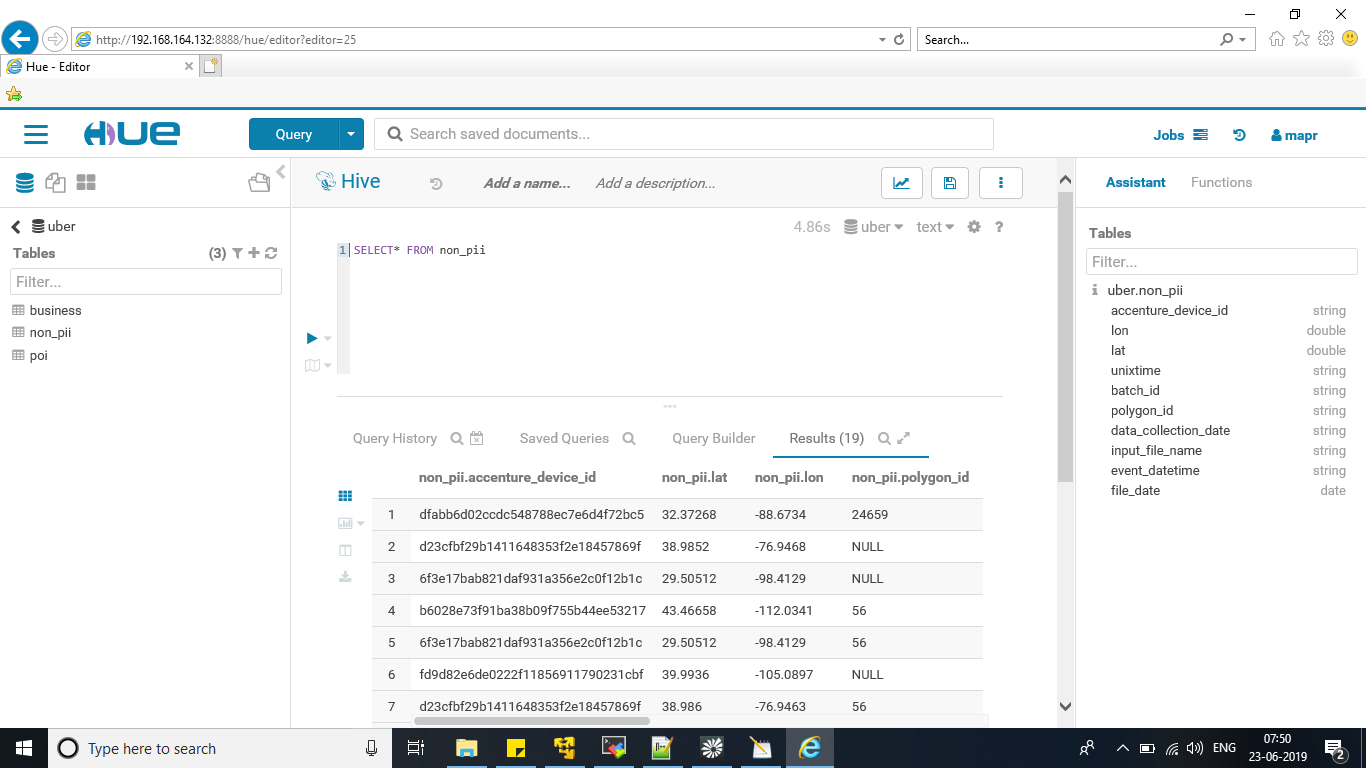


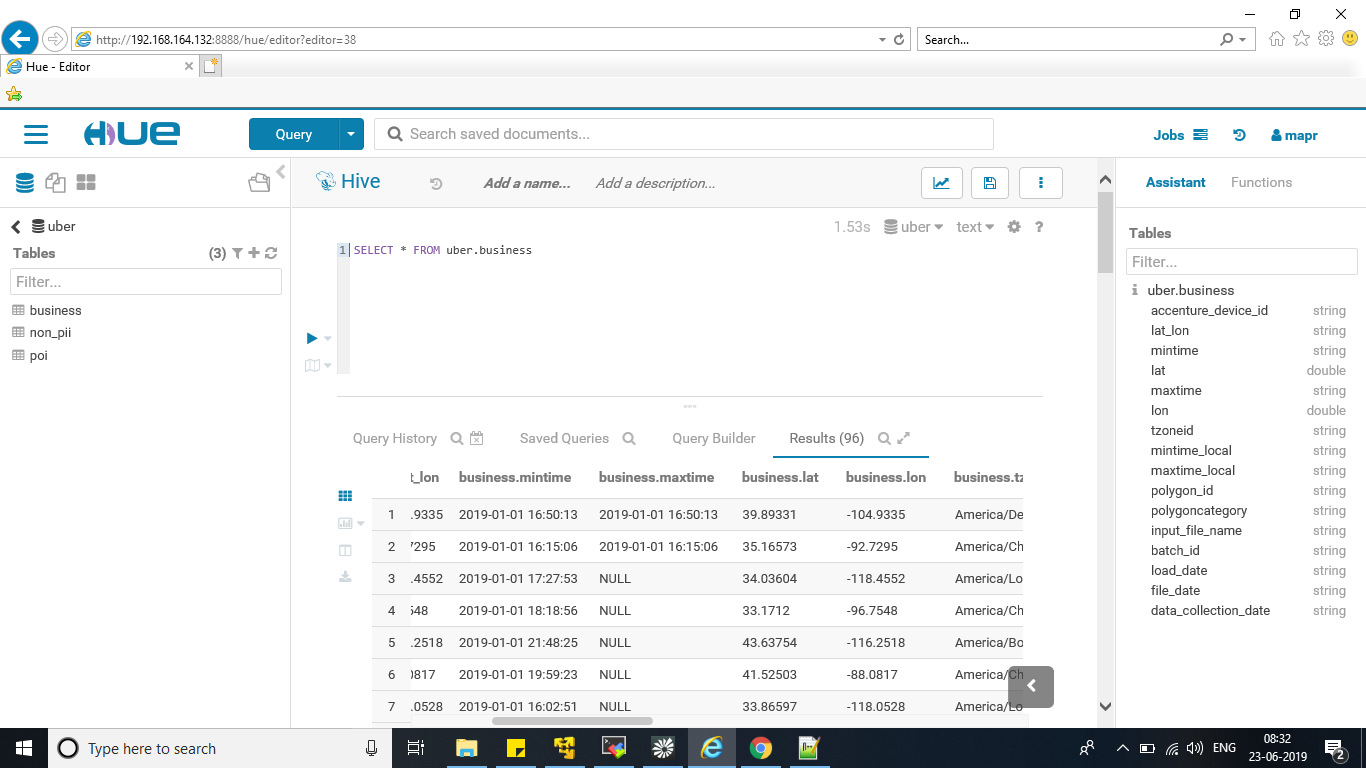


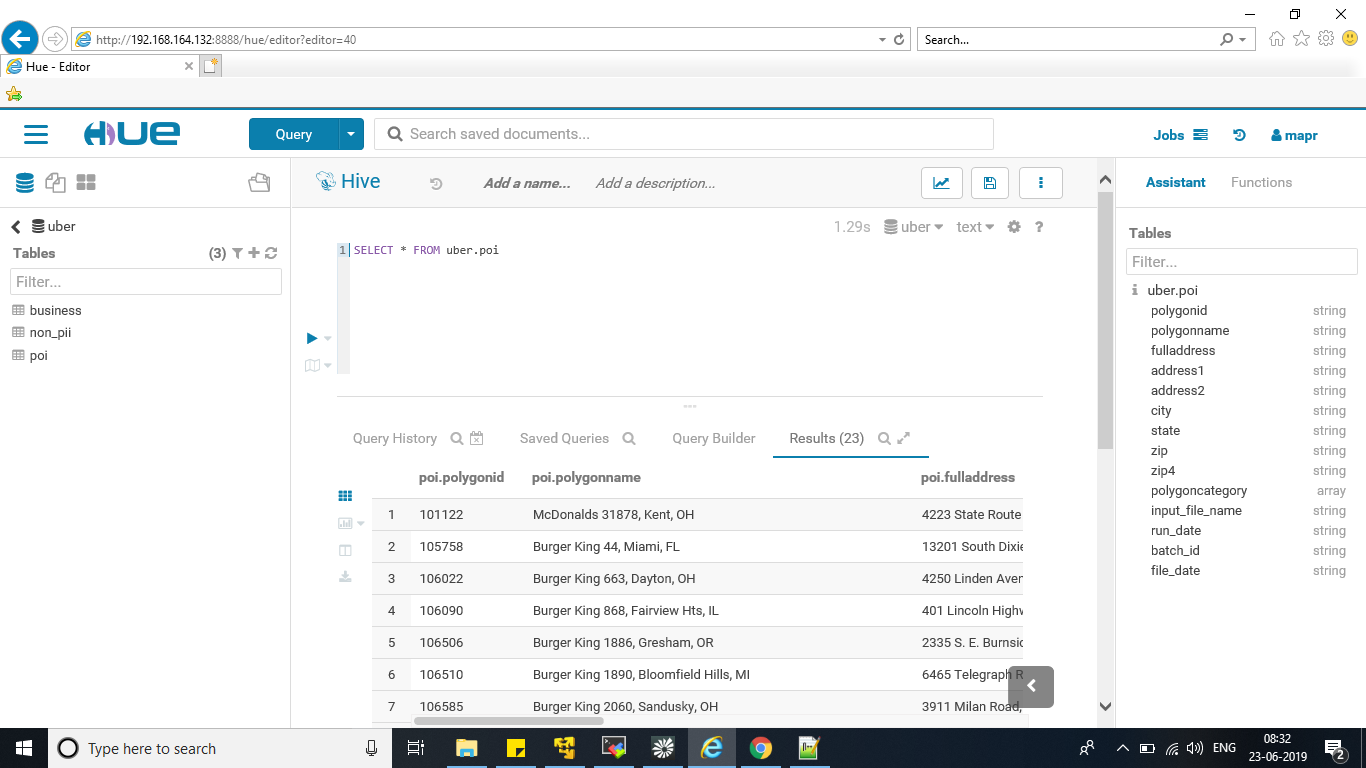


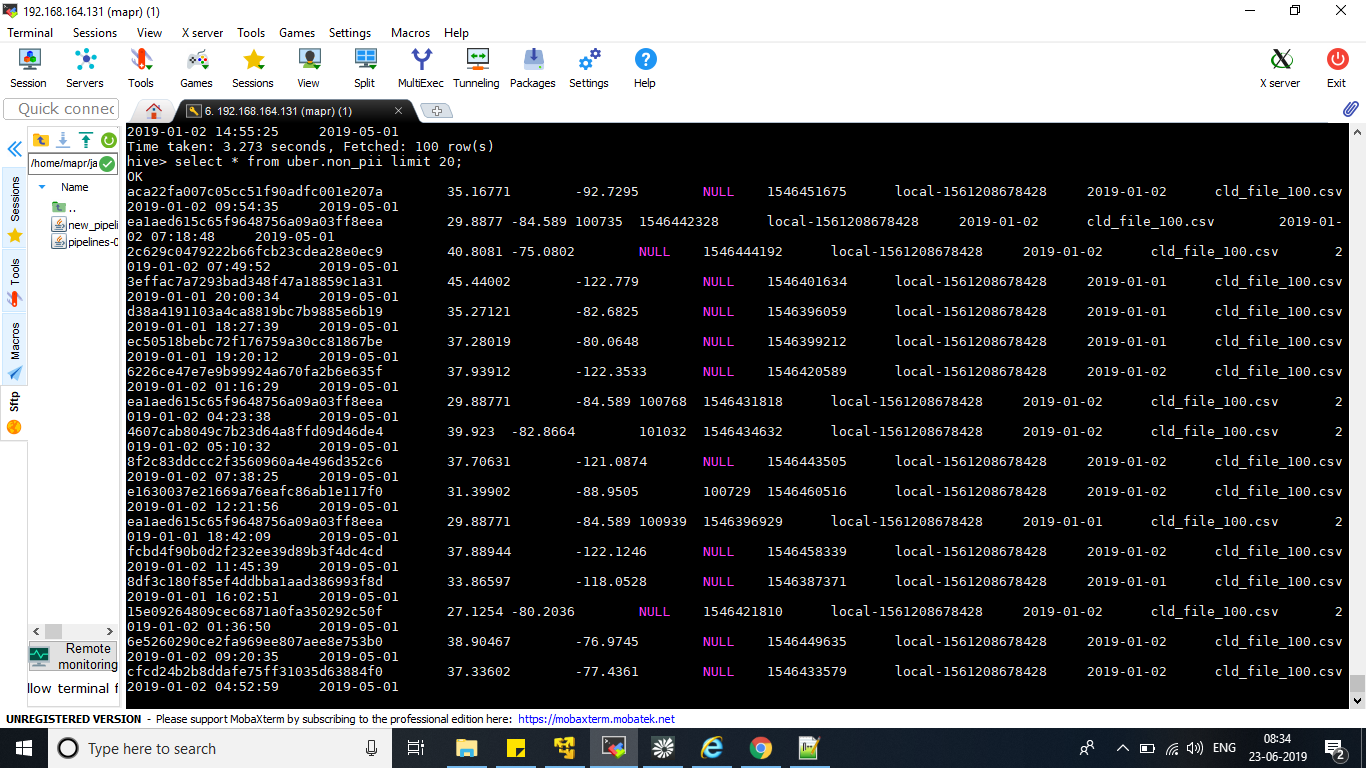


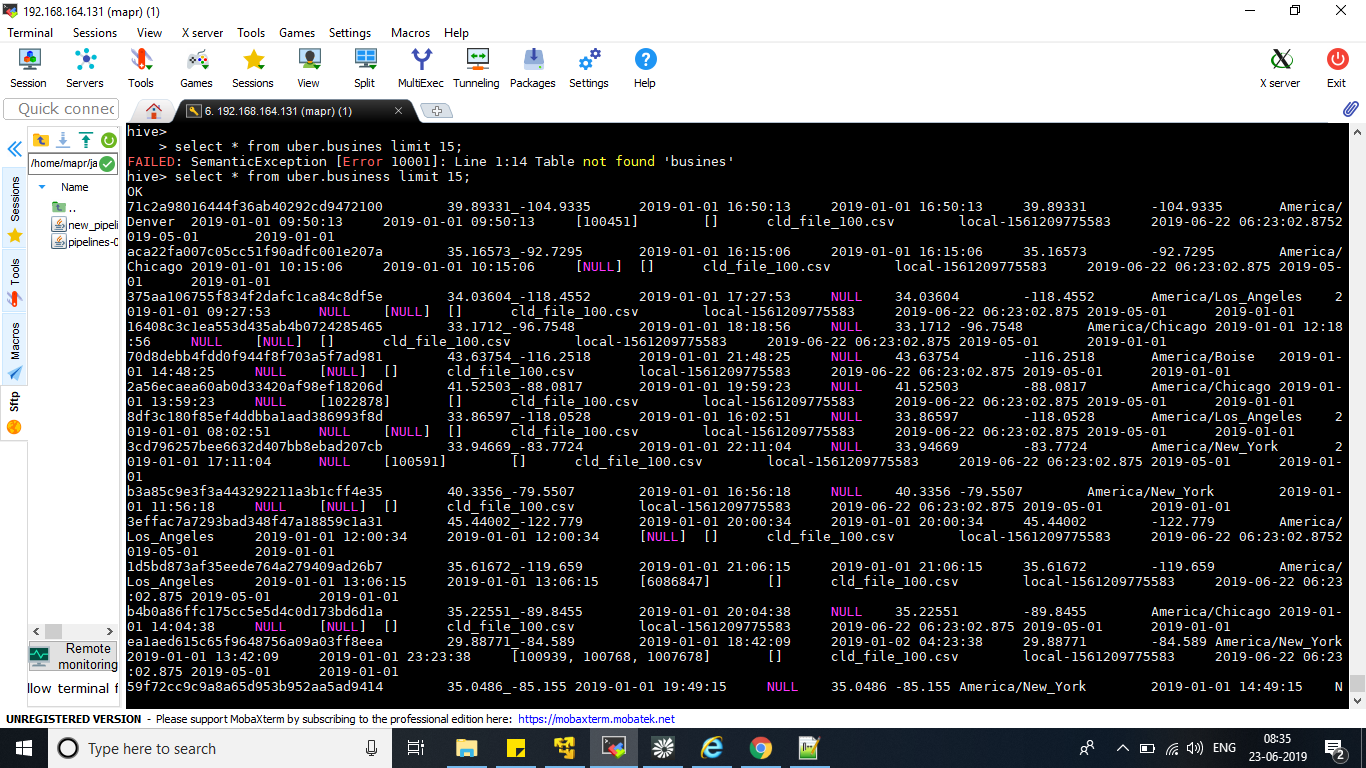


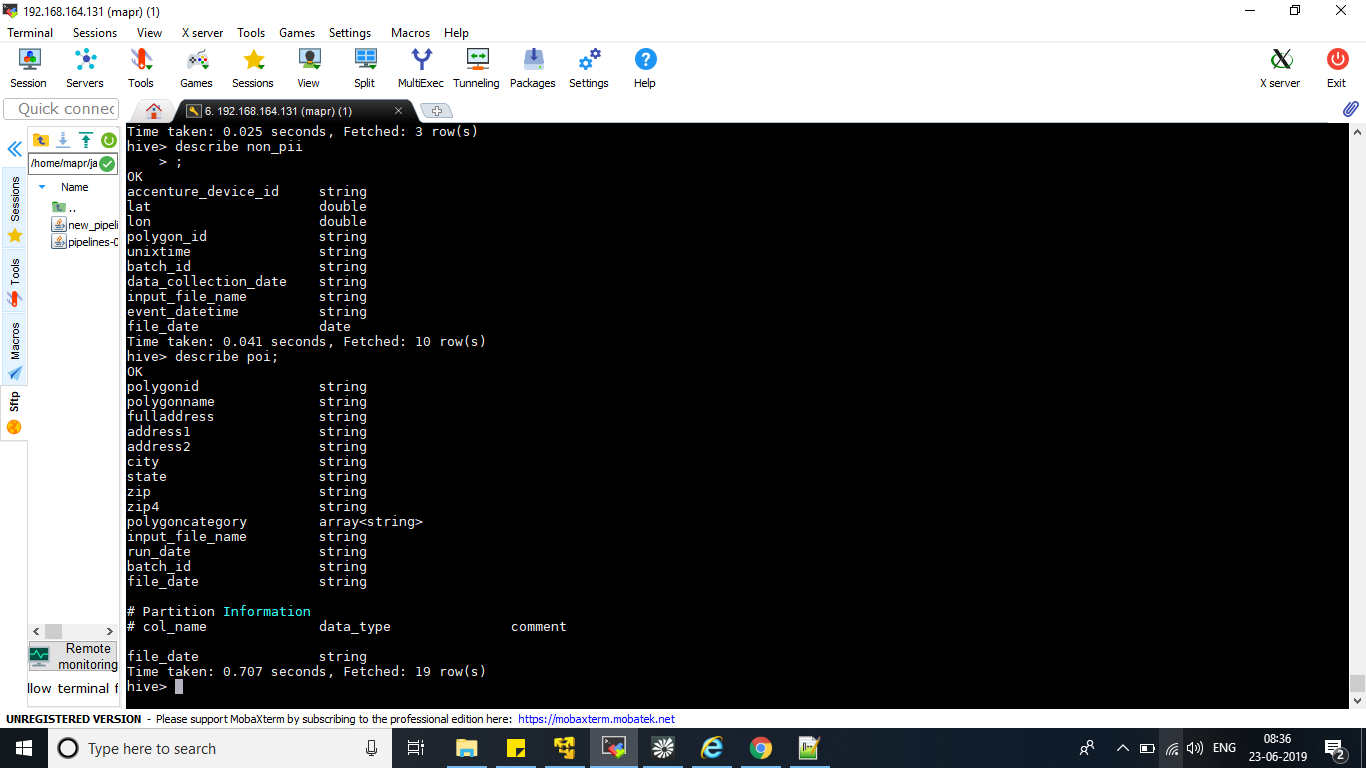


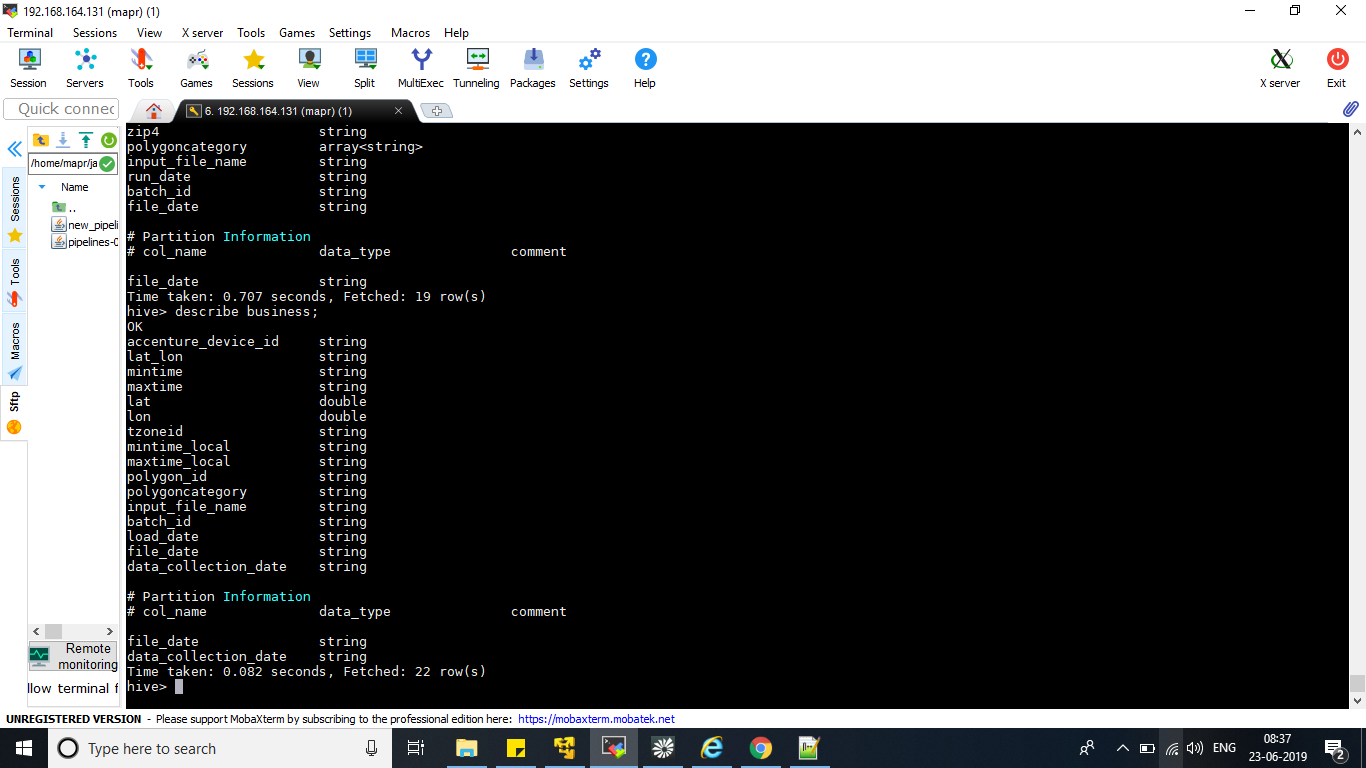












**Conclusion and Future scope of the Project**

Data analysis is a process through we are processing data, cleaning and modified data using tools.

After all those processes completion final result show in the form of good records, using of good data we are making strategies and get desired business results.

This analyzed data we are using for market research

for example, Banking, HealthCare, Government and Education.

Data analysis provide both of them speed and accuracy for business analysis.

Business have realized the importance of utilizing big data to maximize the profit.

All they know this is very important to business purpose and in the sector of healthcare.

As a result, we can say that big data analysis is more important part becoming in the future to making the big economy of the country.

At last if we can say that big data is big buzz in the market, all e-commerce company and social sites like Facebook, Instagram, WhatsApp are using big data technologyHadoop.

**References**

Most of the project I have done in my training time.

1. For this link: -tennysusantobi.blogspot.com
2. For this link: - Submitted to University of Sydney
3. For this link: - forum.freeipodguide.com
4. For this link: - www.ericlin.me

**Appendix**

**Coding Phase: -**

POI Schema

package com.analytic.ubermedia.raw

import com.analytic.ubermedia.raw.ds\_um\_driver.\_

import com.analytic.ubermedia.raw.ds\_um\_utils.\_

import org.apache.spark.sql.{ Encoders, SparkSession }

import org.apache.spark.storage.StorageLevel.\_

import org.apache.spark.sql.functions.\_

import org.apache.spark.sql.types.IntegerType

import sys.process.\_

// Case class defining POI schema :

case class schema\_pdf(`Polygon ID`: String, `Polygon Name`: String, `Full Address`: String, `Address1`: String, `Address2`: String, `City`: String, `State`: String, `Zip`: BigDecimal, `Zip4`: BigDecimal, `Polygon Category`: String, `bad\_record`: String)

object ds\_um\_poi {

import spark.implicits.\_

def ds\_um\_poi\_func(path\_nm: String, aws\_error\_bucket: String, spark: SparkSession) =

{

var sb: StringBuffer = new StringBuffer(CurrentTime() + “ : Uber Media POI Ingestion process started….” + “\n”)

val path\_name: String = path\_nm

val run\_date: String = spark.sql(“select current\_timestamp() as run\_date”).first().get(0).toString

// Getting job batch id :

val batch\_id: String = app\_id

sb.append(CurrentTime() + “ : spark application ID : “ + batch\_id + “ \n”)

//Extracting folder name from input path and converting it into date :

val filedate = path\_name.split(“/”).last

val file\_date = file\_day(filedate)

val file\_date1 = file\_date.toString

sb.append(CurrentTime() + “ : POI Data is processing for : “ + file\_date1 + “ \n”)

try {

//getting Initial file count :

val total\_file\_count = get\_file\_count(path\_name).toString

sb.append(CurrentTime() + “ : Total number of files present at given Input Path : “ + total\_file\_count + “ \n”)

val total\_file\_count1 = total\_file\_count.toInt

if (total\_file\_count1 == 0) {

throw new CustomException(“ No files are present at Input Path for processing….”)

}

//creating expected array for header comparision :

sb.append(CurrentTime() + “ : creating expected array for header comparisions ….” + “ \n”)

val expec\_arr = Array(“Polygon ID”, “Polygon Name”, “Full Address”, “Address1”, “Address2”, “City”, “State”, “Zip”, “Zip4”, “Polygon Category”)

sb.append(CurrentTime() + “ : Created expected array for header comparision adoopfully….” + “ \n”)

// AWS daily error path creation

sb.append(CurrentTime() + “ : Started creating AWS error path/directory….” + “ \n”)

val aws\_bad\_dest\_path: String = aws\_error\_bucket + “/” + path\_name.split(“/”).last

val aws\_bad\_dest\_path\_dir: String = aws\_bad\_dest\_path.replace(“s3n”, “s3a”).toString()

// (s”adoop fs -mkdir $aws\_bad\_dest\_path\_dir”)!

Val succ\_flag: Int = (s”adoop fs -mkdir $aws\_bad\_dest\_path\_dir”)!

If (succ\_flag != 0) {

throw new CustomException(“ Issue in error directory creation …”)

}

sb.append(CurrentTime() + “ : Created AWS error path/directory adoopfully….” + “ \n”)

// Segregating good and bad files in resp s3 buckets :

sb.append(CurrentTime() + “ : Segregating good and bad files in resp s3 buckets started….” + “ \n”)

df\_file\_segregation(path\_name, expec\_arr, aws\_bad\_dest\_path, spark)

sb.append(CurrentTime() + “ : Segregating good and bad files in resp s3 buckets completed successfully….” + “ \n”)

// Creating Raw schema from case class defined previously :

sb.append(CurrentTime() + “ : Creating custom schema….” + “ \n”)

val RawFileSchema = Encoders.product[schema\_pdf].schema

sb.append(CurrentTime() + “ : Custom schema created ….” + “ \n”)

//Creating Dataframe by reading data from S3 :

sb.append(CurrentTime() + “ : Creating dataframe form\_csv\_df01 by reading data from S3 bucket….” + “ \n”)

val form\_csv\_df01 = spark.read.format(“com.databricks.spark.csv”).option(“header”, “true”)

.option(“columnNameOfCorruptRecord”, “bad\_record”)

.option(“mode”, “PERMISSIVE”)

.schema(RawFileSchema)

.option(“badRecordsPath”, “user/acnhive/umd\_poi\_bad/”)

// .option(“quote”, “\””)

.option(“delimiter”, “\t”)

.load(path\_name)

.withColumn(“input\_file\_name”, get\_last(split(input\_file\_name, “/”)))

.toDF(“PolygonID”, “PolygonName”, “FullAddress”, “Address1”, “Address2”, “City”, “State”, “Zip”, “Zip4”, “PolygonCategory”, “bad\_record”, “input\_file\_name”)

// Calculating Total record count :

sb.append(CurrentTime() + “ : Started calculating Total record\_count ….” + “ \n”)

val record\_count = form\_csv\_df01.count().toString

sb.append(CurrentTime() + “ : Total record\_count is : “ + record\_count + “ \n”)

// Creating dataframe form\_csv\_df\_09 by dropping duplicate records :

sb.append(CurrentTime() + “ : Started creating Dataframe form\_csv\_df\_09 by dropping duplicate records….” + “ \n”)

val form\_csv\_df\_09 = form\_csv\_df01.select($”PolygonID”, $”PolygonName”, $”FullAddress”, $”Address1”, $”Address2”, $”City”, $”State”, $”Zip”.cast(IntegerType), $”Zip4”.cast(IntegerType), $”PolygonCategory”, $”bad\_record”, $”input\_file\_name”).dropDuplicates()

sb.append(CurrentTime() + “ : Created Dataframe form\_csv\_df\_09 by dropping duplicate records adoopfully….” + “ \n”)

// Calculating duplicate record count :

sb.append(CurrentTime() + “ : Started calculating Duplicate records..” + “ \n”)

val duplicate\_record\_count = ((record\_count.toInt) – (form\_csv\_df\_09.count())).toString

sb.append(CurrentTime() + “ : Total count of duplicate records is : “ + duplicate\_record\_count + “ \n”)

// Creating bad record dataframe :

sb.append(CurrentTime() + “ : Started creating bad\_record dataframe on the basis of data type/no.of records in a row….” + “ \n”)

val bad\_records\_df = form\_csv\_df\_09.filter($”bad\_record”.isNotNull).select($”bad\_record”, $”input\_file\_name”).persist(MEMORY\_ONLY)

//Counting bad records Generic :

sb.append(CurrentTime() + “ : Started calculating Bad record count on the basis of data type/no.of records in a row….” + “ \n”)

val bad\_records\_df\_count = bad\_records\_df.count()

sb.append(CurrentTime() + “ : Bad record count on the basis of data type/no.of records in a row : “ + bad\_records\_df\_count + “ \n”)

// Writing bad records to S3 bucket :

sb.append(CurrentTime() + “ : Started writing bad records on the basis of data type/no.of records in a row in S3 bucket…” + “ \n”)

bad\_records\_df.coalesce(1).write.format(“com.databricks.spark.csv”).mode(“append”)

.option(“delimiter”, “\t”).option(“quoteMode”, “ALL”).option(“header”, “false”)

.save(aws\_bad\_dest\_path)

sb.append(CurrentTime() + “ : Bad records on the basis of data type/no.of records in a row written successfully in S3 bucket…” + “ \n”)

// Unpersisting bad record dataframe :

bad\_records\_df.unpersist()

//Creating dataframe by dropping bad records :

val form\_csv\_df02 = form\_csv\_df\_09.filter($”bad\_record” isNull).drop($”bad\_record”)

// Creating dataframe by filtering Null datafields as per adoopf requirement :

sb.append(CurrentTime() + “ : Creating dataframe by filtering Null datafields as per adoopf requirement…..” + “ \n”)

val form\_csv\_df\_bad = form\_csv\_df02.filter(($”PolygonID” isNull) || ($”PolygonName” isNull) || ($”FullAddress” isNull) || ($”PolygonCategory” isNull) || ($”PolygonID” === ““) || ($”PolygonCategory” === ““) || ($”PolygonName” === ““) || ($”FullAddress” === ““))

// Writing null datafields to S3 bucket :

form\_csv\_df\_bad.coalesce(1).write.format(“com.databricks.spark.csv”).mode(“append”)

.option(“delimiter”, “\t”).option(“header”, “false”)

.save(aws\_bad\_dest\_path)

sb.append(CurrentTime() + “ : Completed writing Null datafields as per adoopf requirement in bad records folder to S3 buckets …” + “ \n”)

// Counting record\_error\_count :

sb.append(CurrentTime() + “ : Started calculating total record\_error\_count …” + “ \n”)

val record\_error\_count = (bad\_records\_df\_count + form\_csv\_df\_bad.count()).toString

sb.append(CurrentTime() + “ : Total record\_error\_count is : “ + record\_error\_count + “ \n”)

// Creating dataframe after filter bad and null records :

sb.append(CurrentTime() + “ : Started creating good dataframe form\_csv\_df\_08 after filter bad and null records .. “ + “ \n”)

val form\_csv\_df\_08 = form\_csv\_df02.filter(($”PolygonID” isNotNull) && ($”PolygonName” isNotNull) && ($”PolygonCategory” isNotNull) && ($”FullAddress” isNotNull)).toDF()

val form\_csv\_df\_07 = form\_csv\_df\_08.filter(($”PolygonID” =!= ““) && ($”PolygonName” =!= ““) && ($”PolygonCategory” =!= ““) && ($”FullAddress” =!= ““)).toDF()

// val form\_csv\_df\_06 = form\_csv\_df\_07.filter(($”Zip” isNotNull)||($”Zip4” isNotNull))

val form\_csv\_df = form\_csv\_df\_07.withColumn(“run\_date”, lit(run\_date))

.withColumn(“batch\_id”, lit(batch\_id)).toDF()

//Creating temp table using form\_csv\_df dataframe :

sb.append(CurrentTime() + “ : Started Creating temp table using form\_csv\_df dataframe…...” + “ \n”)

form\_csv\_df.createOrReplaceTempView(“form\_csv\_df\_table”)

//Creating dataframe by collecting polygon categories for resp. polygon id’s and then dropping duplicates :

sb.append(CurrentTime() + “ : Started Creating dataframe by collecting polygon categories for resp. polygon id’s and then dropping duplicates…....” + “ \n”)

val form\_csv\_df\_stg = spark.sql(s”select distinct PolygonID,PolygonName,FullAddress,Address1,Address2,City,State,Zip,Zip4,collect\_set(PolygonCategory) over (PARTITION BY PolygonID) as PolygonCategory,input\_file\_name,run\_date,batch\_id from form\_csv\_df\_table”)

// Calculation Hive record count :

sb.append(CurrentTime() + “ : Started calculating records posted in hive….....” + “ \n”)

val hive\_process\_count = form\_csv\_df\_stg.count().toString

sb.append(CurrentTime() + “ : Total records posted in hive is : “ + hive\_process\_count + “ \n”)

// Creating temp view :

form\_csv\_df\_stg.createOrReplaceTempView(“csv\_df\_02\_table”)

// writing Non-PI data in HDFS

sb.append(CurrentTime() + “ : Started writing Non-PII data in HDFS/HIVE …” + “ \n”)

spark.sql(s”Insert overwrite table ${hive\_table} partition(file\_date=’${file\_date}’) select \* from csv\_df\_02\_table”)

sb.append(CurrentTime() + “ : Completed writing Non-PII data in HDFS/HIVE …” + “ \n”)

// Data Movement Process :

sb.append(CurrentTime() + “ : Archive directory creation process started in s3…..” + “ \n”)

//Creating archival Directory

val aws\_input\_path: String = path\_name.replace(“s3n”, “s3a”).toString()

val aws\_archive\_bucket\_dir: String = (aws\_archive\_bucket + “/” + path\_name.split(“/”).last).replace(“s3n”, “s3a”).toString()

val arch\_flag: Int = (s”adoop fs -mkdir $aws\_archive\_bucket\_dir”)!

If (arch\_flag != 0) {

throw new CustomException(“Issue in archive directory creation ..”)

}

sb.append(CurrentTime() + “ : Archive directory created in s3 and file movement process started…..” + “ \n”)

//File movement started:

val mv\_flag: Int = (s”adoop fs -mv $aws\_input\_path/\* $aws\_archive\_bucket\_dir “)!

If (mv\_flag != 0) {

throw new CustomException(“ Issue in file movement in archive directory ..”)

}

sb.append(CurrentTime() + “ : Files moved adoopfully in archival directory and input directory removal process started…..” + “ \n”)

//Removing Input Folder :

val rm\_flag: Int = (s”adoop fs -rmdir $aws\_input\_path”)!

If (rm\_flag != 0) {

throw new CustomException(“ Issue in removing Input folder after files movement in archive folder..”)

}

sb.append(CurrentTime() + “ : Input folder deleted successfully from s3…..” + “ \n”)

// Getting total file processed count :

sb.append(CurrentTime() + “ : Total files processed count started….....” + “ \n”)

val process\_file\_count = get\_file\_count(aws\_archive\_bucket\_dir.replace(“s3a”, “s3n”).toString()).toString

sb.append(CurrentTime() + “ : Total file processed count is : “ + process\_file\_count + “ \n”)

// Getting Error file count :

sb.append(CurrentTime() + “ : Error file count process started….....” + “ \n”)

val error\_file\_count = ((total\_file\_count.toInt) – (process\_file\_count.toInt)).toString

sb.append(CurrentTime() + “ : Total error file count is : “ + error\_file\_count + “ \n”)

// Audit Process started :

sb.append(CurrentTime() + “ : Audit process started…...” + “ \n”)

audit\_table\_queries(run\_date, file\_type, “processed”, path\_name, batch\_id, total\_file\_count, process\_file\_count, error\_file\_count, record\_count, duplicate\_record\_count, hive\_process\_count, record\_error\_count, file\_date1)

sb.append(CurrentTime() + “ : Audit process completed successfully…..” + “ \n”)

} //Exception Handling started :

catch {

// Generic Exception :

case e: Exception =>

e.printStackTrace

sb.append(CurrentTime() + “ : Audit process failed; Normal Exception raised; Exception cause ->“ + e.getMessage + “ \n”)

for (element <- e.getStackTrace) { sb.append(element.toString) }

sb.append(“\n”)

audit\_table\_queries(run\_date, file\_type, “failed”, path\_name, batch\_id, “0”, “0”, “0”, “0”, “0”, “0”, “0”, file\_date1)

// Custom Exception :

case c: CustomException =>

c.printStackTrace

sb.append(CurrentTime() + “ : Audit process failed; Custom Exception raised; Custom Exception cause ->“ + c.getMessage + “ \n”)

for (element <- c.getStackTrace) { sb.append(element.toString) }

sb.append(“\n”)

audit\_table\_queries(run\_date, file\_type, “failed”, path\_name, batch\_id, “0”, “0”, “0”, “0”, “0”, “0”, “0”, file\_date1)

} //Finally Block Started :

finally {

val logwrite = sb.toString

val df = Seq(logwrite).toDF().coalesce(1)

df.write.mode(“append”).csv(log\_path + “/” + file\_date)

}

}

}

**Packages**

package com.analytic.ubermedia.raw

import com.analytic.ubermedia.raw.ds\_um\_driver.\_

import org.apache.spark.sql.SparkSession

import org.apache.hadoop.fs.\_

import org.apache.spark.sql.functions.\_

import scala.collection.mutable.ListBuffer

import java.text.SimpleDateFormat

import java.sql.Timestamp

object ds\_um\_utils {

// function takes the input\_path as argument and returns filenames present at that path :

def get\_file\_name(input\_path: String): Set[String] = {

val src = new Path(input\_path)

val fs = src.getFileSystem(spark.sparkContext.hadoopConfiguration)

val status = fs.listStatus(src)

var file\_nms = new ListBuffer[String]()

for (x <- status) {

var filename = x.getPath.getName.toString()

file\_nms += filename

}

var file\_name = file\_nms.toSet

file\_name

}

//Function to count number of files present at given location :

def get\_file\_count(file\_path: String): Int =

{

val src = new Path(file\_path)

val fs = src.getFileSystem(spark.sparkContext.hadoopConfiguration)

val file\_cnt = fs.getContentSummary(src).getFileCount().toInt

file\_cnt

}

//Function to segregate files in good and bad folder in S3 for processing :

def df\_file\_segregation(input\_path: String, expec\_arr: Array[String], aws\_bad\_dest\_path: String, spark: SparkSession): Unit =

{

val src = new Path(input\_path)

val fs = src.getFileSystem(spark.sparkContext.hadoopConfiguration)

val path\_name: String = input\_path

var file\_name = get\_file\_name(path\_name)

for (x <- file\_name) {

val fnm: String = x

val src\_file\_path: String = path\_name + “/” + fnm

val raw\_csv\_arr = spark.read.format(“com.databricks.spark.csv”).option(“header”, “true”)

.option(“codec”, “org.apache.hadoop.io.compress.Bzip2Codec”).option(“delimiter”, “\t”)

.load(src\_file\_path).columns

val val\_check: Boolean = expec\_arr.sameElements(raw\_csv\_arr)

if (val\_check.equals(false)) {

val aws\_bad\_file\_path = new Path(src\_file\_path)

val aws\_bad\_dest1 = new Path(aws\_bad\_dest\_path)

fs.rename(aws\_bad\_file\_path, aws\_bad\_dest1)

}

}

}

// Function to get date from CEL and POI folder names :

def file\_day(t1: String): String =

{

val format: SimpleDateFormat = new SimpleDateFormat(“yyyyMM”) //Input format given

val formatter: SimpleDateFormat = new SimpleDateFormat(“yyyy-MM-dd”) //Output format as required

val d = format.parse(t1)

val day = formatter.format(d)

return day

}

// Function to get date from CLD folder name :

def file\_day\_cld(t1: String): String =

{

val format: SimpleDateFormat = new SimpleDateFormat(“ddMMMyyyy”) //Input format given

val formatter: SimpleDateFormat = new SimpleDateFormat(“yyyy-MM-dd”) //Output format as required

val d = format.parse(t1)

val day = formatter.format(d)

return day

}

//Method to get current timestamp for logger:

def CurrentTime(): String =

{

var d: Timestamp = new Timestamp(System.currentTimeMillis())

return d.toString()

}

// Application Id calculation :

val app\_id: String = spark.sparkContext.applicationId

// Segregating file name from filepath :

val get\_last = udf((xs: Seq[String]) => xs.last)

// Audit table processing :

def audit\_table\_queries(run\_date: String, file\_type: String, status: String, src\_path: String, app\_id: String, total\_file\_count: String, process\_file\_count: String, error\_file\_count: String, record\_count: String, duplicate\_record\_count: String, hive\_process\_count: String, record\_error\_count: String, file\_date1: String) =

{

//val audit\_df = spark.sql(s”select ‘${run\_date}’ as run\_date,’${file\_type}’ as file\_type,’${status}’ as status,’${src\_path}’ as input\_file\_path,’${app\_id}’ as batch\_id,’${total\_file\_count}’ as total\_file\_count,’${process\_file\_count}’ as process\_file\_count,’${error\_file\_count}’ as error\_file\_count,’${record\_count}’as records\_sent\_inProcessing,’${duplicate\_record\_count}’ as duplicate\_record\_count,’${hive\_process\_count}’ as records\_posted\_inHive,’${record\_error\_count}’ as records\_posted\_inError”)

val audit\_df = spark.sql(s”select ‘${run\_date}’ as run\_date,’${file\_type}’ as file\_type,’${status}’ as status,’${src\_path}’ as input\_file\_path,’${app\_id}’ as batch\_id,’${total\_file\_count}’ as total\_file\_count,’${process\_file\_count}’ as process\_file\_count,’${error\_file\_count}’ as error\_file\_count,’${record\_count}’ as total\_records\_count,’${duplicate\_record\_count}’ as duplicate\_record\_count,’${hive\_process\_count}’ as records\_posted\_inHive,’${record\_error\_count}’ as records\_posted\_inError,’${file\_date1}’ as file\_date”)

audit\_df.coalesce(1).write.format(“com.databricks.spark.csv”).mode(“append”).option(“delimiter”, “,”).option(“header”, “false”).save(aws\_audit\_table)

}

}

**Driver Class**

package com.analytic.ubermedia.raw

import ds\_um\_contextual\_loc.\_

import ds\_um\_evening\_loc.\_

import ds\_um\_poi.\_

import org.apache.log4j.{Level, Logger}

import org.apache.spark.sql.{Row, SaveMode, SparkSession}

import org.apache.hadoop.fs.\_

import org.apache.hadoop.conf.Configuration

//Declaring final case class for using custom exceptions:

final case class CustomException(private val message: String = “”,private val cause: Throwable = None.orNull) extends Exception(message, cause)

object ds\_um\_driver extends App

{

// Creating Spark Session :

val spark = SparkSession

.builder()

.appName(s”${this.getClass.getSimpleName}”)

//.master(“local[\*]”)

.master(“local[\*]”)

// .config(“spark.scheduler.mode”, “FAIR”)

// .config(“spark.serializer”,”org.apache.spark.serializer.KryoSerializer”)

// .config(“spark.io.compression.codec”,”lz4”)

.enableHiveSupport()

.getOrCreate()

// val logger: Logger = Logger.getLogger(“ds\_um\_driver”)

// logger.info(“INFO”)

// Setting logger level for log file creation:

spark.sparkContext.setLogLevel(“INFO”)

// Setting Hadoop Configruation

val hadoopConfig = new Configuration()

val hdfs = FileSystem.get(hadoopConfig)

// Configruing AWS Credentials for Generic ACNHIVE user :

spark.sparkContext.hadoopConfiguration.set(“fs.s3n.impl”, “org.apache.hadoop.fs.s3native.NativeS3FileSystem”)

spark.sparkContext.hadoopConfiguration.set(“fs.s3n.awsAccessKeyId”, “AKIAJRKYNIBHHKPJI5WA”)

spark.sparkContext.hadoopConfiguration.set(“fs.s3n.awsSecretAccessKey”, “5NlozFu6tGd/jg7VxT1gUPvXViJHDgCai1+Oetf/”)

// Setting runtime parameters need to be called in job :

// file-type is basically job name :

val file\_type = args(0).toString()

//Input file path :

val input\_path = args(1).toString()

// redshift temp directory :

// val tempS3Dir = args(2).toString()

//Raw hive table location :

val hive\_table = args(2).toString()

// S3 process and error bucket :

val aws\_error\_bucket = args(3).toString()

val aws\_archive\_bucket = args(4).toString()

//audit table path :

val aws\_audit\_table = args(5).toString()

//redshift table : “hadoopschema.ubermedia\_deviceid”

val pii\_table = args(6).toString()

//log path :

val log\_path = args(7).toString

//Job Type : QA or PROD

// val job\_type = args(9).toString

// Checking for which job type whether QA or PROD need to make REDSHIFT connection :

/\*val jdbcURL = if(job\_type == “PROD”){“jdbc:redshift://adsuedscaired02.cohfy3ssrkcy.us-east-1.redshift.amazonaws.com:5439/ds\_proddb?user=prodhadoopuser&password=Prodhadoop2pass”}

else {“jdbc:redshift://adsuedscaired02.cohfy3ssrkcy.us-east-1.redshift.amazonaws.com:5439/ds\_qadb?user=qahadoopuser&password=Qahadoop2pass”}

\*/

// Calling Functions as per the requirement :

if (file\_type == “UM\_DAILY\_CONTEXTUAL\_LOCATION”)

{

ds\_um\_contextual\_loc\_func(input\_path,aws\_error\_bucket,spark)

}

else if (file\_type == “UM\_MONTHLY\_EVENING\_LOCATION”) {

ds\_um\_evening\_loc\_func(input\_path,aws\_error\_bucket,spark)

}

else if (file\_type == “UM\_POI”){

ds\_um\_poi\_func(input\_path,aws\_error\_bucket,spark)

}

else {

println(“Job Failed Due to Invalid File Type Argument”)

System.exit(1)

}

}

**Contextual Local Data**

package com.analytic.ubermedia.raw

import com.analytic.ubermedia.raw.ds\_um\_driver.\_

import com.analytic.ubermedia.raw.ds\_um\_utils.\_

import org.apache.spark.sql.{Encoders, SparkSession}

import org.apache.spark.storage.StorageLevel.\_

import org.apache.spark.sql.functions.\_

import sys.process.\_

// Case class defining CLD schema :

case class schema\_cdf(device\_id:String,unixtime:Long,lat:BigDecimal,lon:BigDecimal,polygon\_id:String,bad\_record:String)

object ds\_um\_contextual\_loc {

import spark.implicits.\_

def ds\_um\_contextual\_loc\_func(path\_nm:String,aws\_error\_bucket:String,spark: SparkSession) =

{

var sb: StringBuffer = new StringBuffer(CurrentTime() +” : Uber Media CLD Raw Layer Ingestion process started….”+”\n”)

val path\_name:String = path\_nm

val run\_date: String = spark.sql(“select current\_timestamp() as run\_date”).first().get(0).toString

val redshift\_run\_date: String = spark.sql(“select current\_date() as run\_date”).first().get(0).toString

// Getting job batch id :

val batch\_id: String = app\_id

sb.append(CurrentTime() +” : spark application ID : “+batch\_id+ “ \n”)

//Extracting folder name from input path and converting it into date :

val filedate = path\_name.split(“/”).last

val file\_date = file\_day\_cld(filedate)

val file\_date1 = file\_date.toString

sb.append(CurrentTime() +” : CLD Raw Data is processing for : “+file\_date1 + “ \n”)

try

{

//Getting Initial total file count :

val total\_file\_count = get\_file\_count(path\_name).toString

sb.append(CurrentTime() +” : Total number of files present at given Input Path : “+total\_file\_count+ “ \n”)

var total\_file\_count1 = total\_file\_count.toInt

if(total\_file\_count1 == 0)

{

throw new CustomException(“ No files are present at Input Path for processing…”)

}

//Creating expected array for header comparision :

sb.append(CurrentTime() +” : Started creating expected array for header comparision….”+ “ \n”)

val expec\_arr = Array(“device\_id”,”unixtime”,”lat”,”lon”,”polygon\_id”)

sb.append(CurrentTime() +” : Created expected array for header comparision adoopfully….”+ “ \n”)

// AWS daily error path creation

sb.append(CurrentTime() +” : Started creating AWS error path/directory….”+ “ \n”)

val aws\_bad\_dest\_path:String = aws\_error\_bucket+”/”+path\_name.split(“/”).last

val aws\_bad\_dest\_path\_dir:String = aws\_bad\_dest\_path.replace(“s3n”,”s3a”).toString()

// (s”adoop fs -mkdir $aws\_bad\_dest\_path\_dir”)!

Val succ\_flag:Int = (s”adoop fs -mkdir $aws\_bad\_dest\_path\_dir”)!

If(succ\_flag != 0)

{

throw new CustomException(“ Issue in error directory creation …”)

}

sb.append(CurrentTime() +” : Created AWS error path/directory adoopfully….”+ “ \n”)

// Segregating good and bad files in resp s3 buckets :

sb.append(CurrentTime() +” : Segregating good and bad files with resp. to headers from resp. s3 buckets started….”+ “ \n”)

df\_file\_segregation(path\_name,expec\_arr,aws\_bad\_dest\_path,spark)

sb.append(CurrentTime() +” : Segregating good and bad files with resp. to headers from resp. s3 buckets completed successfully….”+ “ \n”)

// Creating Raw schema from case class defined previously :

sb.append(CurrentTime() +” : Creating custom schema….”+ “ \n”)

val RawFileSchema = Encoders.product[schema\_cdf].schema

sb.append(CurrentTime() +” : Custom schema created adoopfully….”+ “ \n”)

//Creating Dataframe by reading data from S3 :

sb.append(CurrentTime() +” : Started creating dataframe form\_csv\_df01 by reading data from S3 bucket….”+ “ \n”)

val form\_csv\_df01 = spark.read.format(“com.databricks.spark.csv”).option(“header”, “true”)

.option(“columnNameOfCorruptRecord”,”bad\_record”)

.option(“mode”, “PERMISSIVE”)

.schema(RawFileSchema)

.option(“badRecordsPath”, “user/acnhive/ubermedia/error”)

.option(“delimiter”,”\t”)

.load(path\_name)

.withColumn(“input\_file\_name”,get\_last(split(input\_file\_name,”/”)))

// Calculating Total record count :

sb.append(CurrentTime() +” : Started calculating Total record\_count ….”+” \n”)

val record\_count = form\_csv\_df01.count().toString

sb.append(CurrentTime() +” : Total record\_count is : “+record\_count+ “ \n”)

// Creating dataframe from unique records by rounding lat to 5 decimal places and lon to 4 decimal places :

sb.append(CurrentTime() +” : Started creating Dataframe form\_csv\_df\_09 from unique records by rounding lat to 5 decimal places and lon to 4 decimal places….”+” \n”)

val form\_csv\_df\_09 = form\_csv\_df01.selectExpr(“device\_id”,”unixtime”,”round(lat,5) as lat”,”round(lon,4) as lon”,”polygon\_id”,”bad\_record”,”input\_file\_name”).dropDuplicates()

form\_csv\_df\_09.persist(MEMORY\_ONLY\_SER)

sb.append(CurrentTime() +” : Created Dataframe form\_csv\_df\_09 from unique records by rounding lat to 5 decimal places and lon to 4 decimal places adoopfully….”+” \n”)

// Calculating duplicate record count :

sb.append(CurrentTime() +” : Started calculating Duplicate records..”+” \n”)

val duplicate\_record\_count = ((record\_count.toLong)-(form\_csv\_df\_09.count())).toString

sb.append(CurrentTime() +” : Total count of duplicate records is : “+duplicate\_record\_count+ “ \n”)

// Creating bad record dataframe :

sb.append(CurrentTime() +” : Started creating bad\_record dataframe on the basis of data type/no.of records in a row….”+” \n”)

val bad\_records\_df = form\_csv\_df\_09.filter($”bad\_record”.isNotNull).select($”bad\_record”,$”input\_file\_name”).persist(MEMORY\_ONLY)

//Counting bad records Generic :

sb.append(CurrentTime() +” : Started calculating Bad record count on the basis of data type/no.of records in a row….”+” \n”)

val bad\_records\_df\_count = bad\_records\_df.count()

sb.append(CurrentTime() +” : Bad record count on the basis of data type/no.of records in a row : “+bad\_records\_df\_count+ “ \n”)

// Writing bad records to S3 bucket :

sb.append(CurrentTime() +” : Started writing bad records on the basis of data type/no.of records in a row in S3 bucket…”+ “ \n”)

bad\_records\_df.coalesce(1).write.format(“com.databricks.spark.csv”).mode(“append”)

.option(“delimiter”, “\t”).option(“header”, “false”)

.save(aws\_bad\_dest\_path)

sb.append(CurrentTime() +” : Bad records on the basis of data type/no.of records in a row written successfully in S3 bucket…”+ “ \n”)

// Unpersisting bad record dataframe :

bad\_records\_df.unpersist()

//Creating dataframe by dropping bad records :

// val form\_csv\_df02 = form\_csv\_df01.drop($”bad\_record”).na.drop(“all”)

val form\_csv\_df02 = form\_csv\_df\_09.filter($”bad\_record” isNull).drop($”bad\_record”)

// Creating dataframe by filtering Null datafields as per adoopf requirement :

sb.append(CurrentTime() +” : Creating dataframe by filtering Null datafields as per adoopf requirement…..”+ “ \n”)

val form\_csv\_df\_bad = form\_csv\_df02.filter(($”device\_id” isNull)||($”unixtime” isNull)||($”lat” isNull)||($”lon” isNull)||($”device\_id” === ““)||($”unixtime” === ““)||($”lat” === ““)||($”lon” === ““))

// Writing null datafields to S3 bucket :

sb.append(CurrentTime() +” : Started writing Null datafields as per adoopf requirement in bad records folder to S3 buckets …”+ “ \n”)

form\_csv\_df\_bad.coalesce(1).write.format(“com.databricks.spark.csv”).mode(“append”)

//.option(“codec”, “org.apache.hadoop.io.compress.Lz4Codec”)

.option(“delimiter”, “\t”).option(“header”, “false”)

.save(aws\_bad\_dest\_path)

sb.append(CurrentTime() +” : Completed writing Null datafields as per adoopf requirement in bad records folder to S3 buckets …”+ “ \n”)

// Counting record\_error\_count :

sb.append(CurrentTime() +” : Started calculating total record\_error\_count …”+ “ \n”)

val record\_error\_count = (bad\_records\_df\_count+form\_csv\_df\_bad.count()).toString

sb.append(CurrentTime() +” : Total record\_error\_count is : “+record\_error\_count+ “ \n”)

// Creating dataframe after filter bad and null records :

sb.append(CurrentTime() +” : Started creating good dataframe form\_csv\_df\_08 after filter bad and null records .. “+ “ \n”)

val form\_csv\_df\_08 = form\_csv\_df02.filter(($”device\_id” isNotNull)&&($”unixtime” isNotNull)&&($”lat” isNotNull)&&($”lon” isNotNull)).toDF()

val form\_csv\_df\_07 = form\_csv\_df\_08.filter(($”device\_id” =!= ““)||($”unixtime” =!= ““)||($”lat” =!= ““)||($”lon” =!= ““)).toDF()

//val form\_csv\_df\_07 = form\_csv\_df\_08.filter(trim($”device\_id” =!= “”)).filter(trim($”unixtime” =!= “”)).filter(trim($”lat” =!= “”)).filter(trim($”lon” =!= “”)).toDF()

val form\_csv\_df = form\_csv\_df\_07.withColumn(“adoopfu\_device\_id”, md5($”device\_id”))

.withColumn(“data\_collection\_date”,from\_unixtime($”unixtime”,”yyyy-MM-dd”))

.withColumn(“event\_datetime”,from\_unixtime($”unixtime”,”yyyy-MM-dd HH:mm:ss”))

.withColumn(“batch\_id”, lit(batch\_id))

.withColumn(“file\_date”, lit(file\_date)).toDF()

// Calculation Hive record count :

sb.append(CurrentTime() +” : Started calculating records posted in hive….....”+ “ \n”)

val hive\_process\_count = form\_csv\_df.count().toString

sb.append(CurrentTime() +” : Total records posted in hive is : “+hive\_process\_count+ “ \n”)

// writing PII data device\_id in redshift

sb.append(CurrentTime() +” : Started writing PII data in redshift …”+ “ \n”)

form\_csv\_df.select($”device\_id”.as(“deviceid”), $”adoopfu\_device\_id”, lit(file\_type).as(“file\_type”),lit(redshift\_run\_date).as(“run\_date”)).dropDuplicates()

.write.format(“com.databricks.spark.csv”).mode(“append”).save(pii\_table)

sb.append(CurrentTime() +” : Completed writing PII data in redshift …”+ “ \n”)

// writing Non-PI data in HDFS

sb.append(CurrentTime() +” : Started writing Non-PII data in HDFS…...”+ “ \n”)

form\_csv\_df.select($”adoopfu\_device\_id”, $”lat”, $”lon”, $”polygon\_id”, $”unixtime”,$”batch\_id”, $”data\_collection\_date”,$”input\_file\_name”,$”event\_datetime”,$”file\_date”)

.write.format(“com.databricks.spark.csv”).mode(“overwrite”)

// .option(“codec”, “org.apache.hadoop.io.compress.Lz4Codec”)

.option(“delimiter”, “\t”).option(“header”, “false”)

.save(hive\_table)

sb.append(CurrentTime() +” : Completed writing Non-PII data in HDFS…..”+ “ \n”)

// Data Movement Process :

sb.append(CurrentTime() +” : Archive directory creation process started in s3…..”+ “ \n”)

//Creating archival Directory

val aws\_input\_path:String = path\_name.replace(“s3n”,”s3a”).toString()

val aws\_archive\_bucket\_dir:String = (aws\_archive\_bucket+”/”+path\_name.split(“/”).last).replace(“s3n”,”s3a”).toString()

val arch\_flag:Int = (s”adoop fs -mkdir $aws\_archive\_bucket\_dir”)!

If(arch\_flag != 0)

{

throw new CustomException(“Issue in archive directory creation ..”)

}

sb.append(CurrentTime() +” : Archive directory created in s3 and file movement process started…..”+ “ \n”)

//File movement started:

val mv\_flag:Int = (s”adoop fs -mv $aws\_input\_path/\* $aws\_archive\_bucket\_dir “)!

If(mv\_flag != 0)

{

throw new CustomException(“ Issue in file movement in archive directory ..”)

}

sb.append(CurrentTime() +” : Files moved adoopfully in archival directory and input directory removal process started…..”+ “ \n”)

//Removing Input Folder :

val rm\_flag:Int = (s”adoop fs -rmdir $aws\_input\_path”)!

If(rm\_flag != 0)

{

throw new CustomException(“ Issue in removing Input folder after files movement in archive folder..”)

}

sb.append(CurrentTime() +” : Input folder deleted successfully from s3…..”+ “ \n”)

// Getting total file processed count :

sb.append(CurrentTime() +” : Total files processed count started….....”+ “ \n”)

val process\_file\_count = get\_file\_count(aws\_archive\_bucket\_dir.replace(“s3a”,”s3n”).toString()).toString

sb.append(CurrentTime() +” : Total file processed count is : “+process\_file\_count+ “ \n”)

// Getting Error file count :

sb.append(CurrentTime() +” : Error file count process started….....”+ “ \n”)

val error\_file\_count = ((total\_file\_count.toInt) – (process\_file\_count.toInt)).toString

sb.append(CurrentTime() +” : Total error file count is : “+error\_file\_count+ “ \n”)

//Unpersisting cached data :

form\_csv\_df\_09.unpersist()

// Audit Process started :

sb.append(CurrentTime() +” : Audit process started…...”+ “ \n”)

//audit\_table\_queries(run\_date,file\_type,”processed”,path\_name,batch\_id,total\_file\_count,process\_file\_count,error\_file\_count,record\_count,duplicate\_record\_count,hive\_process\_count,record\_error\_count)

audit\_table\_queries(run\_date,file\_type,”processed”,path\_name,batch\_id,total\_file\_count,process\_file\_count,error\_file\_count,record\_count,duplicate\_record\_count,hive\_process\_count,record\_error\_count,file\_date1)

sb.append(CurrentTime() +” : Audit process completed successfully…..”+ “ \n”)

}