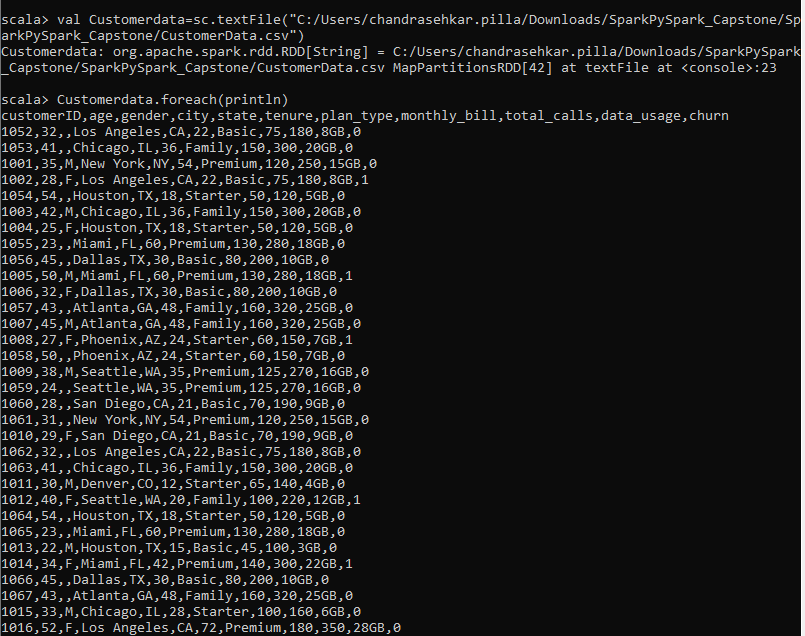
**SPARK CAPSTONE**

**1.Read customer data from HDFS using Spark Core APIs.**

We have to load the data in to the RDD from hdfs directory. Since hdfs is not working we are loading data from local file path.

val Customerdata=sc.textFile("C:/Users/chandrasehkar.pilla/Downloads/SparkPySpark\_Capstone/SparkPySpark\_Capstone/CustomerData.csv")



**2. Implement data cleaning to remove corrupted (CustomerID with ????) and duplicate records.**

We have to remove the duplicate and corrupted data

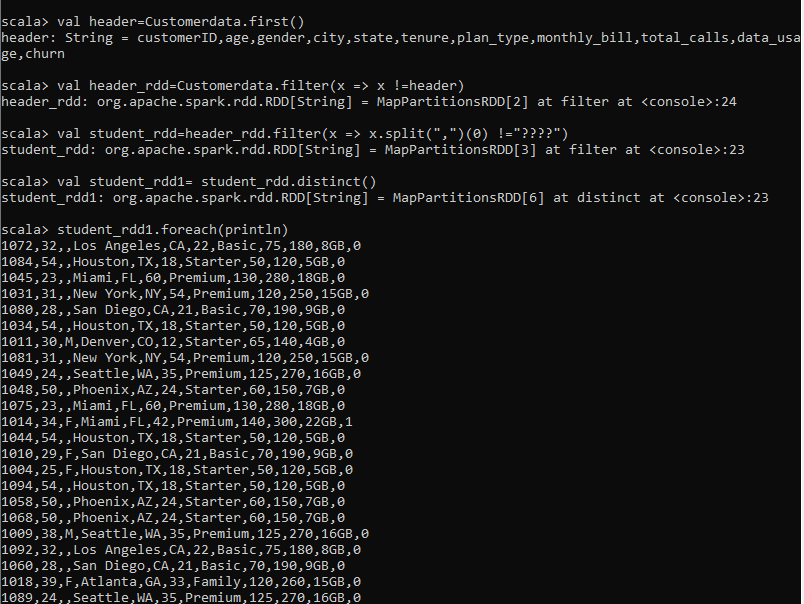
val header=Customerdata.first()

val header\_rdd=Customerdata.filter(x => x !=header)

val student\_rdd=header\_rdd.filter(x => x.split(",")(0) !="????")

val student\_rdd1= student\_rdd.distinct()

student\_rdd1.foreach(println)



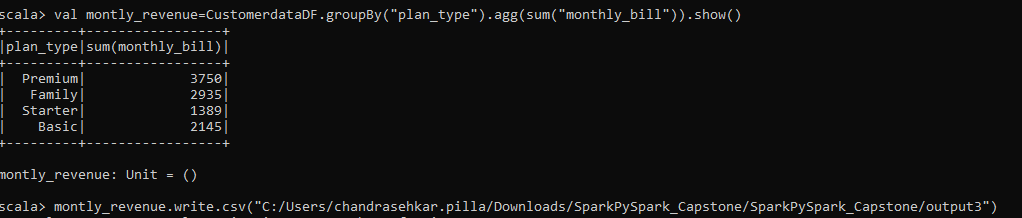
**3. Calculate the monthly revenue generated by different plan types and store the results into HDFS.**

We have to store the data in local path,since hdfs is not working

val CustomerdataDF = student\_rdd1.map(x => x.split(",")).map(c => (c(0).toInt, c(1).toInt, c(2), c(3), c(4), c(5).toInt, c(6), c(7).toInt, c(8).toInt, c(9),c(10))).toDF("customerID","age","gender","city","state","tenure","plan\_type","monthly\_bill","total\_calls","data\_usage","churn")

val montly\_revenue=CustomerdataDF.groupBy("plan\_type").agg(sum("monthly\_bill"))

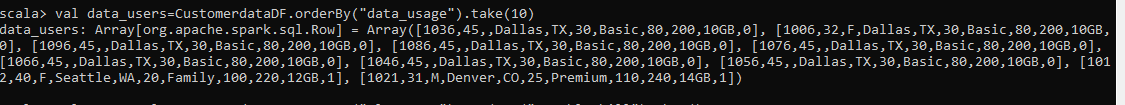
montly\_revenue.write.csv("C:/Users/chandrasehkar.pilla/Downloads/SparkPySpark\_Capstone/SparkPySpark\_Capstone/output3")

****

**4. Identify the top 10 of data users and store the results in a Parquet file for further analysis.**

val data\_users=CustomerdataDF.orderBy("data\_usage").take(10)

data\_users.write.parquet("C:/Users/chandrasehkar.pilla/Downloads/SparkPySpark\_Capstone/SparkPySpark\_Capstone/output2")

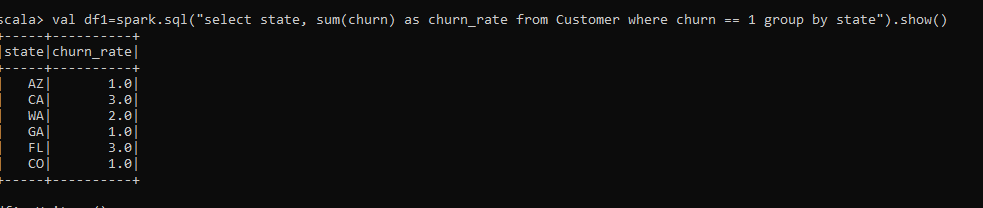
****

**5. Analyze state-wise distribution of churn rate and store the results in a Hive table.**

CustomerdataDF.createOrReplaceTempView("Customer")

val df1=spark.sql("select state, sum(churn) as churn\_rate from Customer where churn == 1 group by state").show()

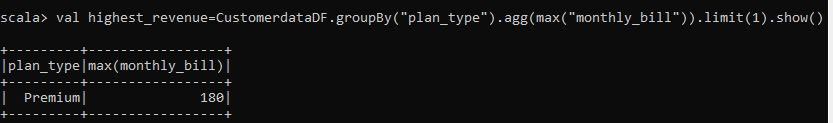
df1.write.option(“path”,”/hive\_table\_location\_path/subpath”).saveAsTable("demodb.Sparkcustomer")

****

**6. Which plan type generates the highest revenue?**

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

val highest\_revenue=CustomerdataDF.groupBy("plan\_type").agg(max("monthly\_bill")).limit(1).show()

****

**7. What is the overall percentage of missing values present in the customer dataset?**

val Customerdata=sc.textFile("C:/Users/chandrasehkar.pilla/Downloads/SparkPySpark\_Capstone/SparkPySpark\_Capstone/CustomerData.csv")

val header=Customerdata.first()

val header\_rdd=Customerdata.filter(x => x !=header)

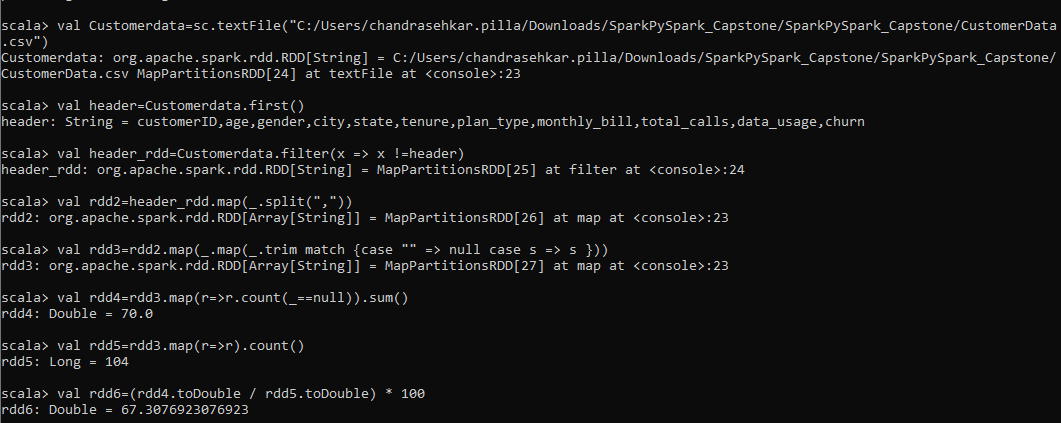
val rdd2=header\_rdd.map(\_.split(","))

val rdd3=rdd2.map(\_.map(\_.trim match {case "" => null case s => s }))

val rdd4=rdd3.map(r=>r.count(\_==null)).sum()

val rdd5=rdd3.map(r=>r).count()

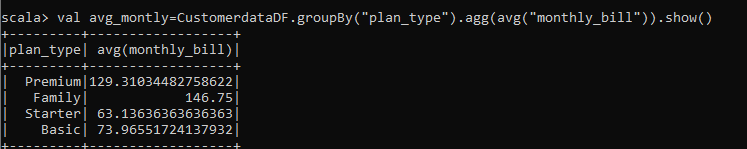
val rdd6=(rdd4.toDouble / rdd5.toDouble) \* 100

****

**8. Calculate the average monthly revenue generated by each customer segment.**

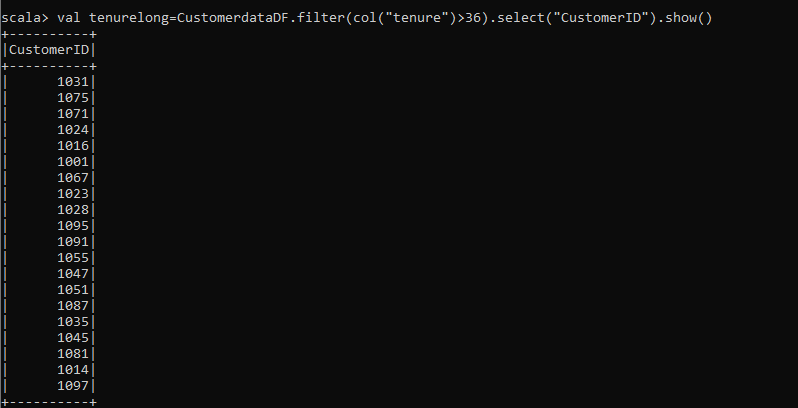
import org.apache.spark.sql.functions.\_

val avg\_montly=CustomerdataDF.groupBy("plan\_type").agg(avg("monthly\_bill")).show()

****

**9. Identify customers with more than 3 years subscription [hint: based on tenure].**

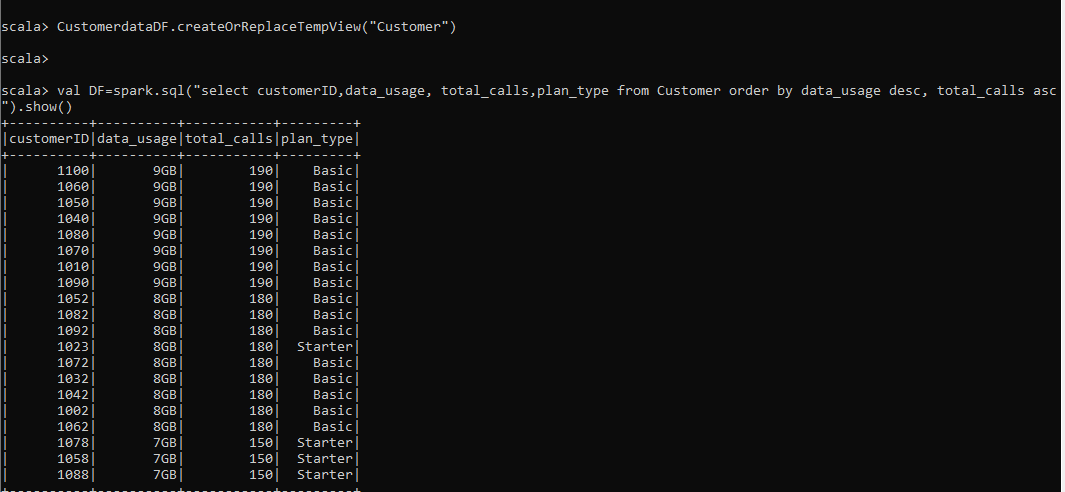
val tenurelong=CustomerdataDF.filter(col("tenure")>36).select("CustomerID").show()

****

**10. Identify customers with high data usage but low call frequency, who might benefit from unlimited data plans.**

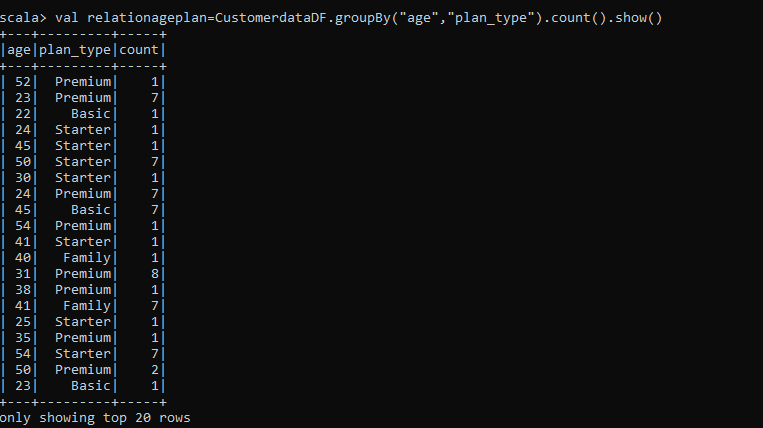
CustomerdataDF.createOrReplaceTempView("Customer")

val DF=spark.sql("select customerID,data\_usage, total\_calls,plan\_type from Customer order by data\_usage desc, total\_calls asc").show()

****

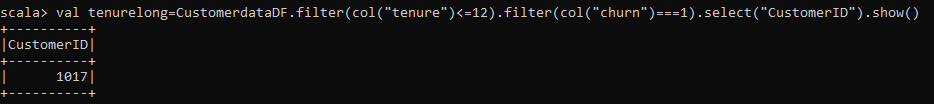
**11. Analyze the correlation between customer age and preferred service plan types.**

val relationageplan=CustomerdataDF.groupBy("age","plan\_type").count().show()

****

**12. Identify customers withdrawn subscription within 12 months.**

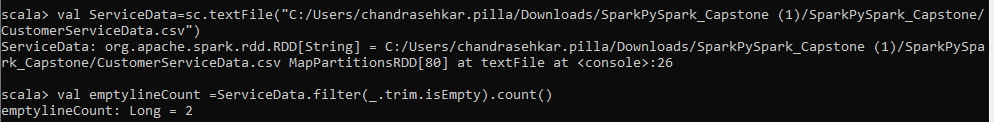
val tenurelong=CustomerdataDF.filter(col("tenure")<=12).filter(col("churn")===1).select("CustomerID").show()

****

**13. Consider customer service dataset and count number of empty lines using accumulators. Remove the empty lines and save the cleaned data as parquet file.**

We have to calculate the empty lines with isempty function.since accumulator is not work.

val emptylineCount =ServiceData.filter(\_.trim.isEmpty).count()

****

val header=ServiceData.first()

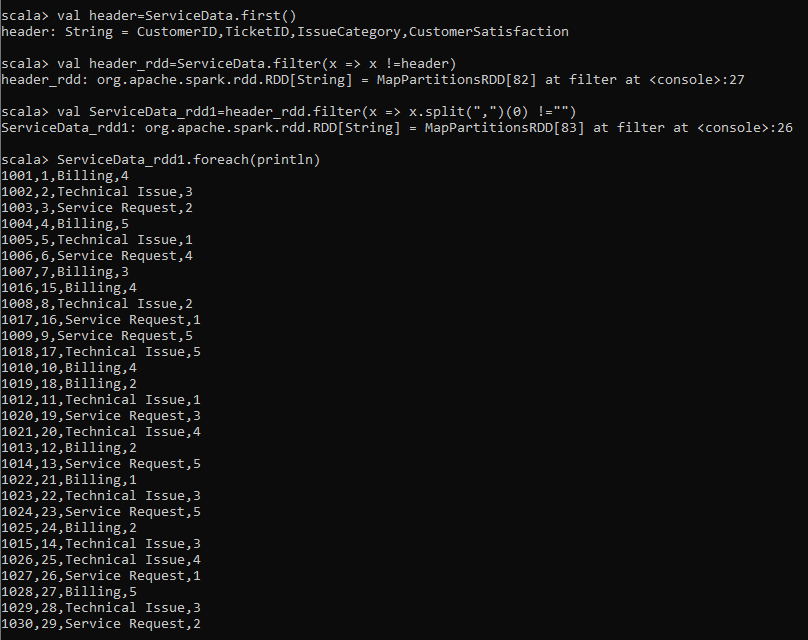
val header\_rdd=ServiceData.filter(x => x !=header)

val ServiceData\_rdd1=header\_rdd.filter(x => x.split(",")(0) !="")

ServiceData\_rdd1.foreach(println)

val ServiceDataDF = ServiceData\_rdd1.map(x => x.split(",")).map(c => (c(0).toInt,c(1).toInt,c(2), c(3).toInt)).toDF("CustomerID","TicketID","IssueCategory","CustomerSatisfaction")

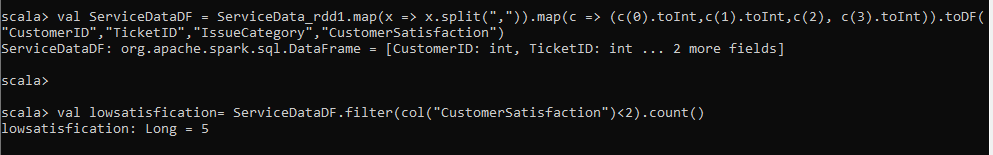
ServiceDataDF.write.parquet("C:/Users/chandrasehkar.pilla/Downloads/SparkPySpark\_Capstone/SparkPySpark\_Capstone/output4")

****

**14. Create a DataFrame from cleaned customer service data and find out the count of service tickets with low satisfaction levels less than 2.**

val ServiceDataDF = ServiceData\_rdd1.map(x => x.split(",")).map(c => (c(0).toInt,c(1).toInt,c(2), c(3).toInt)).toDF("CustomerID","TicketID","IssueCategory","CustomerSatisfaction")

val lowsatisfication= ServiceDataDF.filter(col("CustomerSatisfaction")<2).count()

****

**15. Broadcast customer service data and use it as look-up while processing large customers data. Processed final DataFrame should contain the fields CustomerID, Plan\_Type, Churn, TicketID, IssueCategory,Customer Satisfaction.**

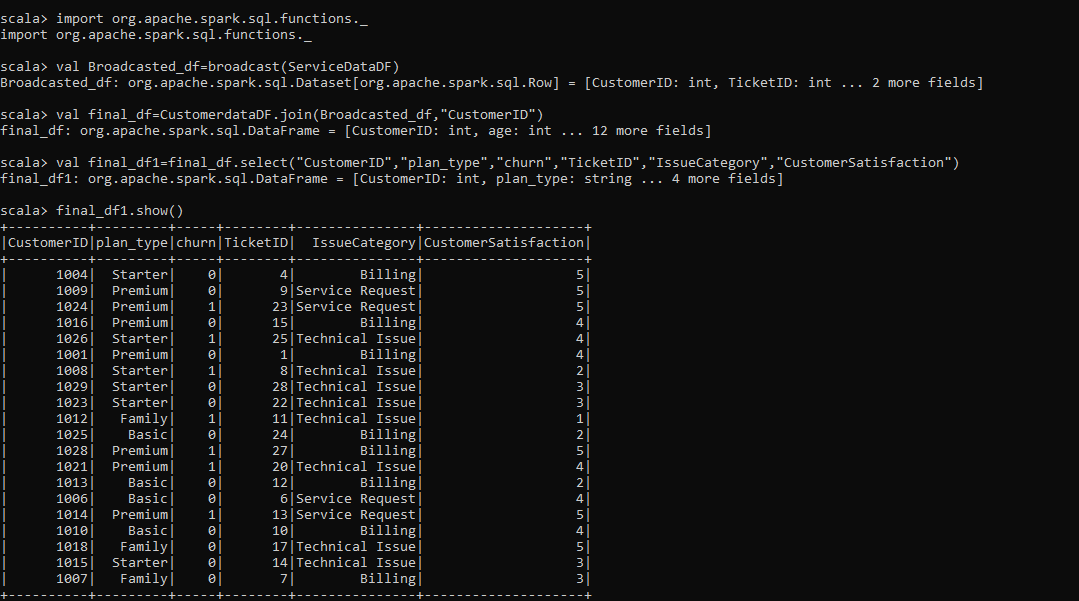
import org.apache.spark.sql.functions.\_

val Broadcasted\_df=broadcast(ServiceDataDF)

val final\_df=CustomerdataDF.join(Broadcasted\_df,"CustomerID")

val final\_df1=final\_df.select("CustomerID","plan\_type","churn","TicketID","IssueCategory","CustomerSatisfaction")

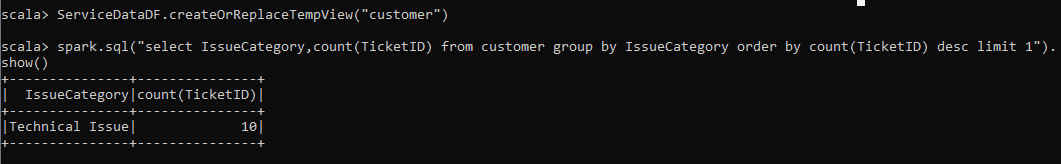
final\_df1.show()



**16. From the above DataFrame, analyze on which IssueCategory most tickets are raised and the reason for churn.**

ServiceDataDF.createOrReplaceTempView("customer")

spark.sql("select IssueCategory,count(TicketID) from customer group by IssueCategory order by count(TicketID) desc limit 1").show()

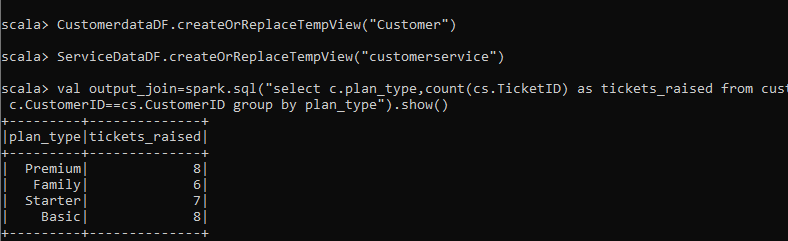
****

**17. Create two DataFrames and Implement Join operation between Customer data and Service data. Calculate the number of tickets raised for each plan\_type.**

CustomerdataDF.createOrReplaceTempView("Customer")

ServiceDataDF.createOrReplaceTempView("customerservice")

val output\_join=spark.sql("select c.plan\_type,count(cs.TicketID) as tickets\_raised from customer c join customerservice cs on c.CustomerID==cs.CustomerID group by plan\_type").show()

****