|  |  |
| --- | --- |
| **IMG_256** | **D:\Big Data Analytics - IIT\Semester 1\logo.pnglogo** |

**INFORMATICS INSTITUTE OF TECHNOLOGY**

**In Collaboration with**

**ROBERT GORDON UNIVERSITY ABERDEEN**

School of Computing Science and Digital Media

MSc Big Data Analytics

2019/2020

By

Niroshan Kumarasinghe

IIT No: 2019269

RGU No: 1912833

CMM705 – Big Data Programming

Coursework

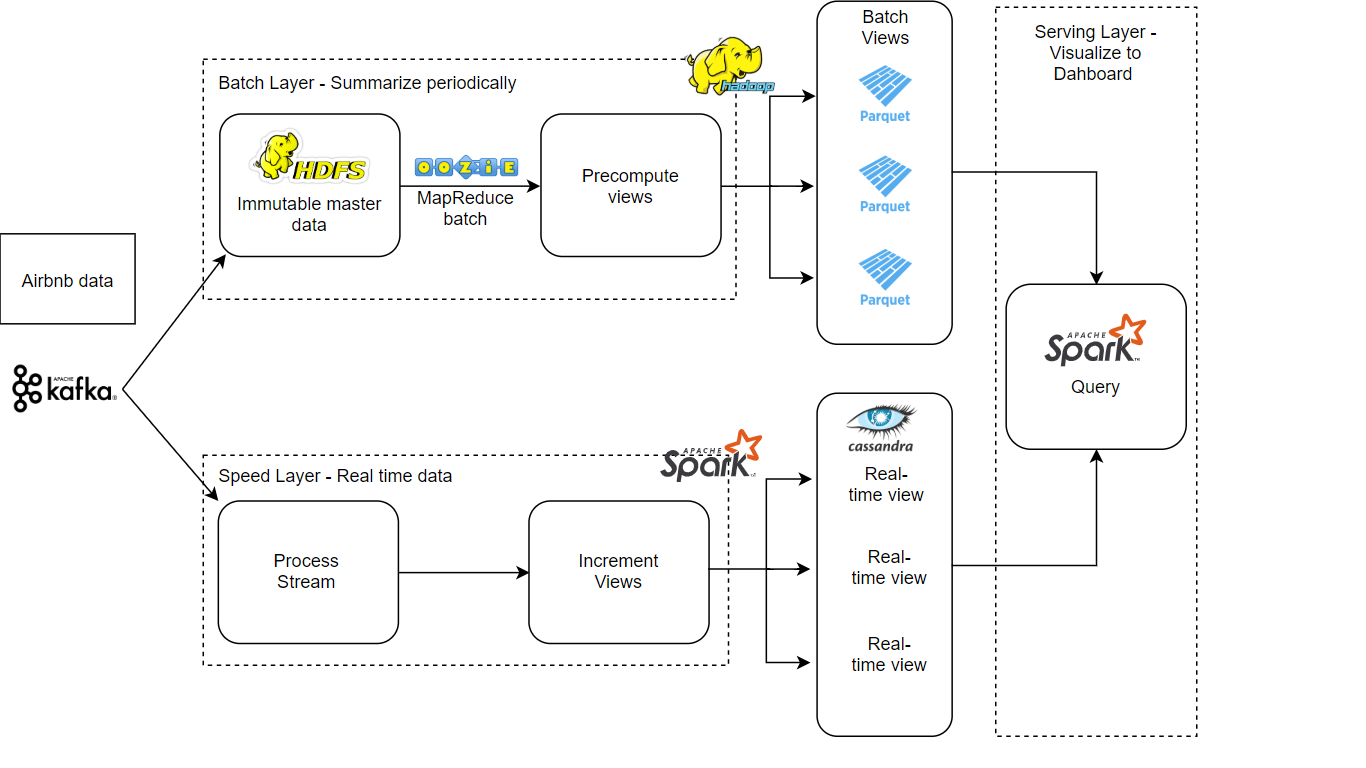
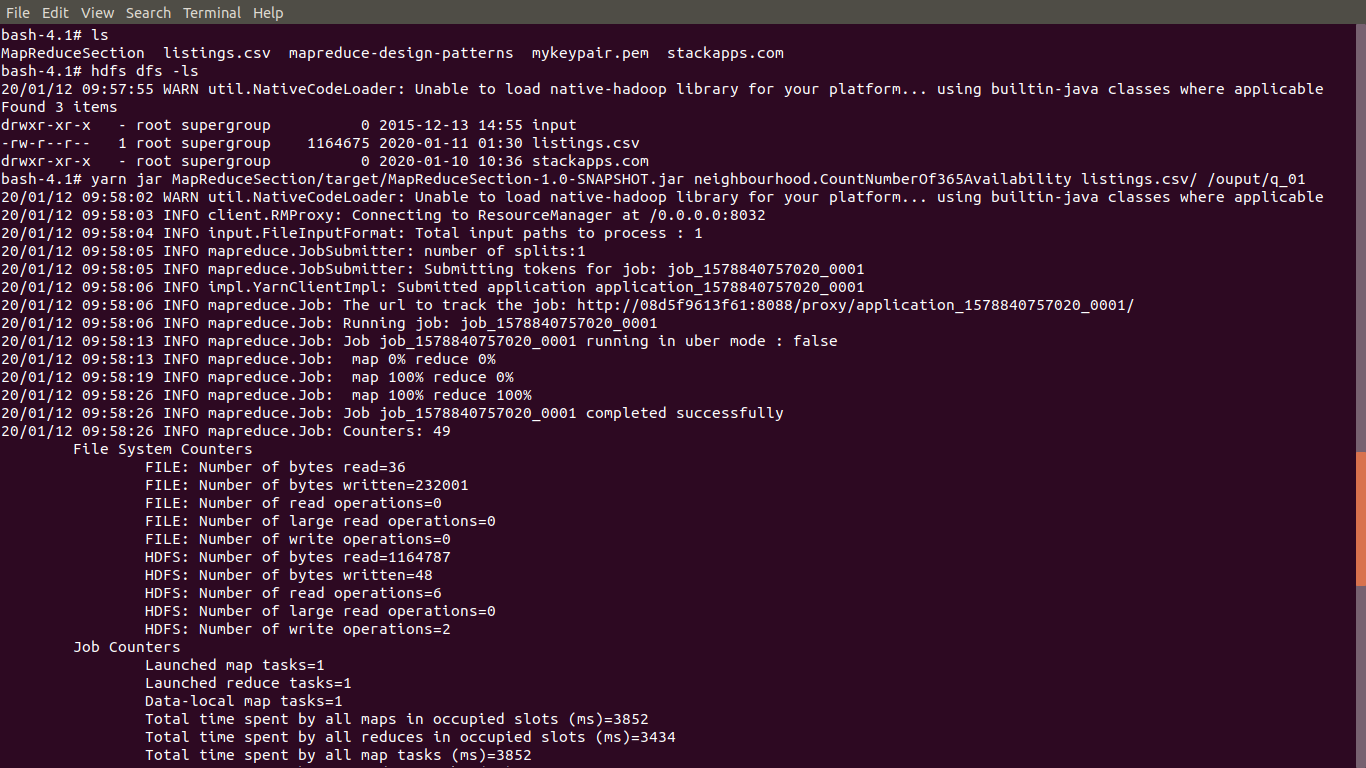
1. **Part One - Deployment Architecture**

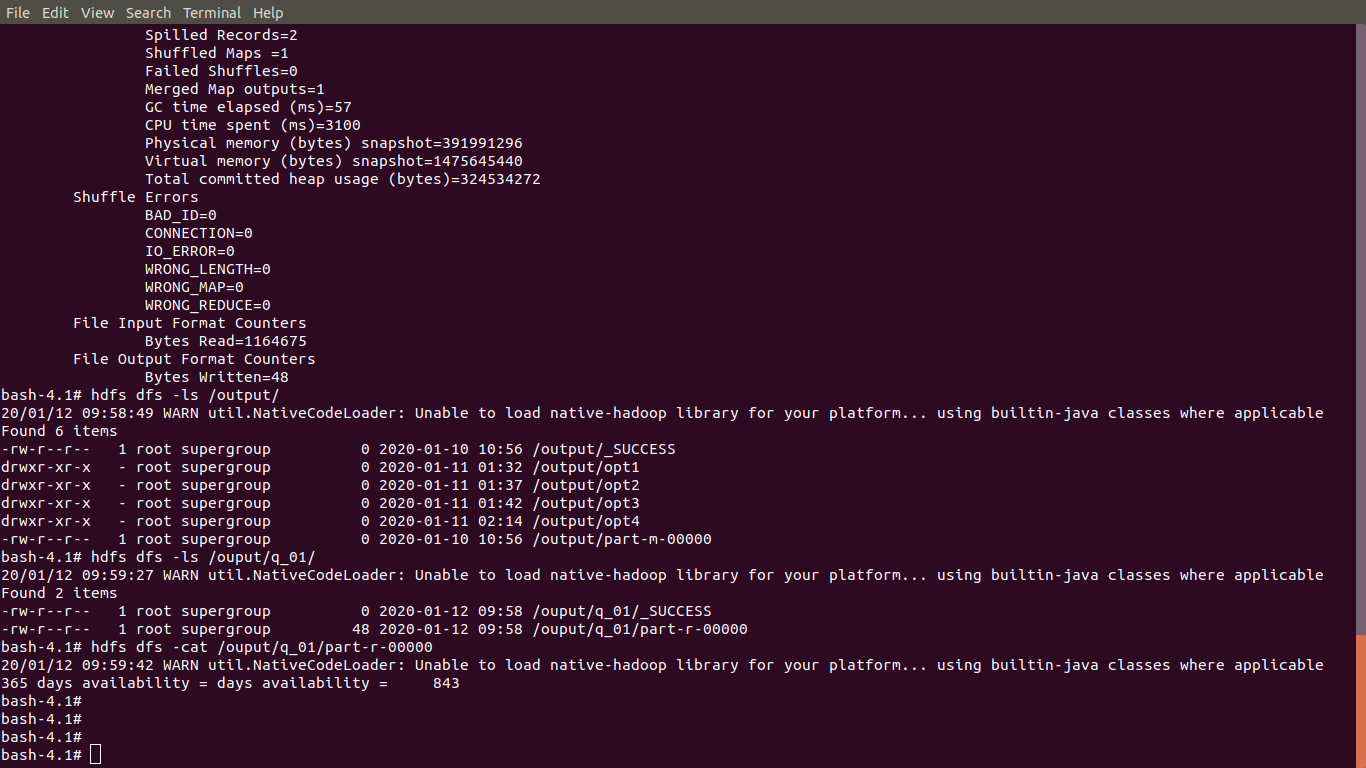
Figure 1: Deployment Diagram (Question 01)

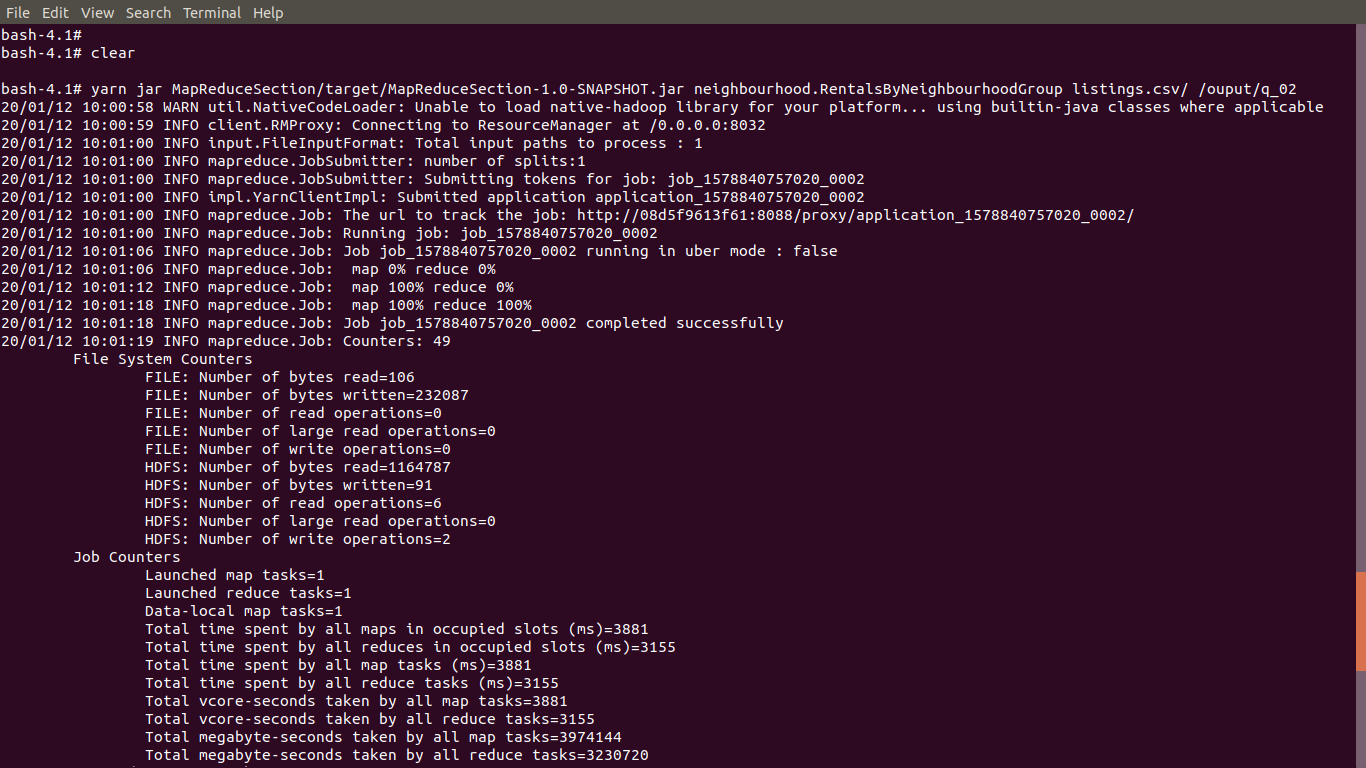
|  |  |
| --- | --- |
| **Tool** | **Implementation** |
| Kafka | Feeding data from Airbnb data sources |
| HDFS | Store immutable master data (archive) |
| Oozie | Manage data and Schedule workflow |
| Spark | Process streams for real-time views |
| Cassandra | Store real-time view (hot path) |
| Parquet | Create batch view and store |
| Spark | Serve to dashboard |

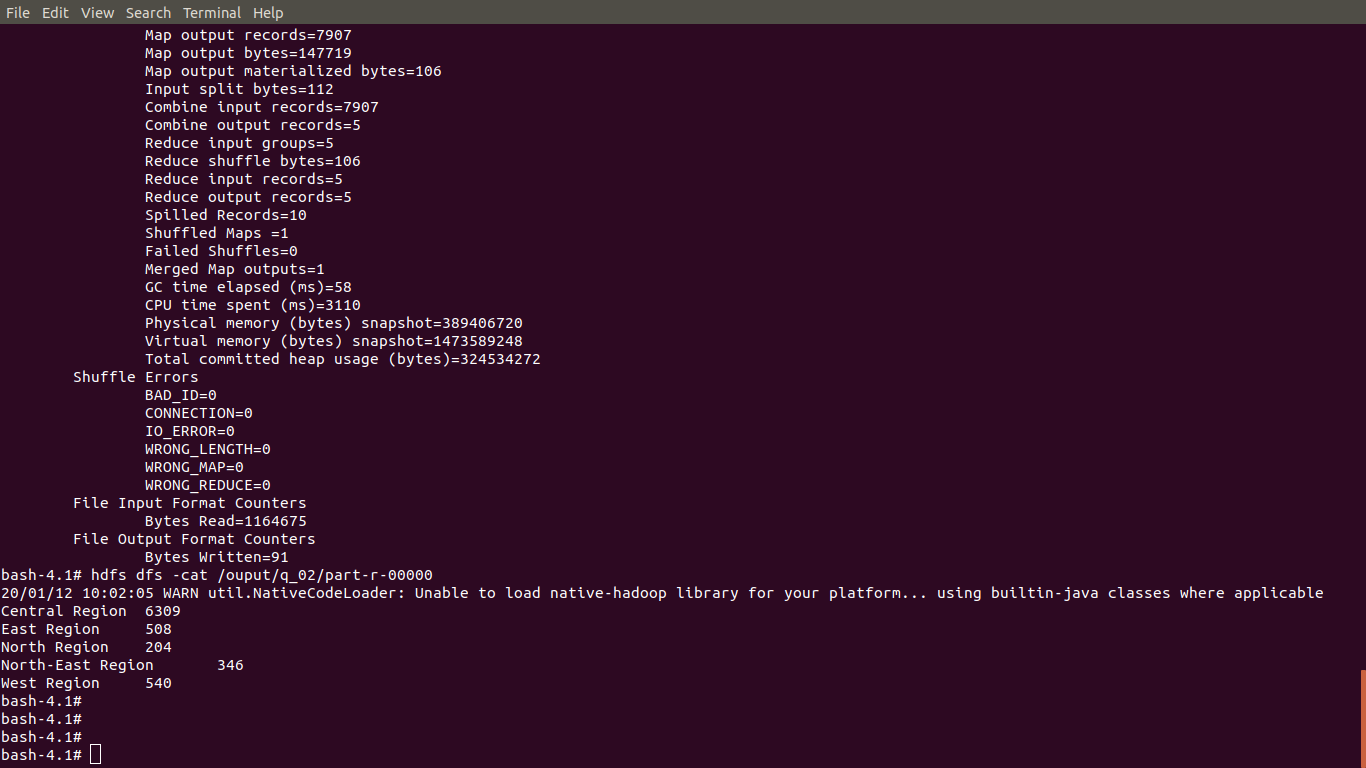
1. **Map reduce jobs/queries and results**

**2.1. Hadoop Map Reduce**

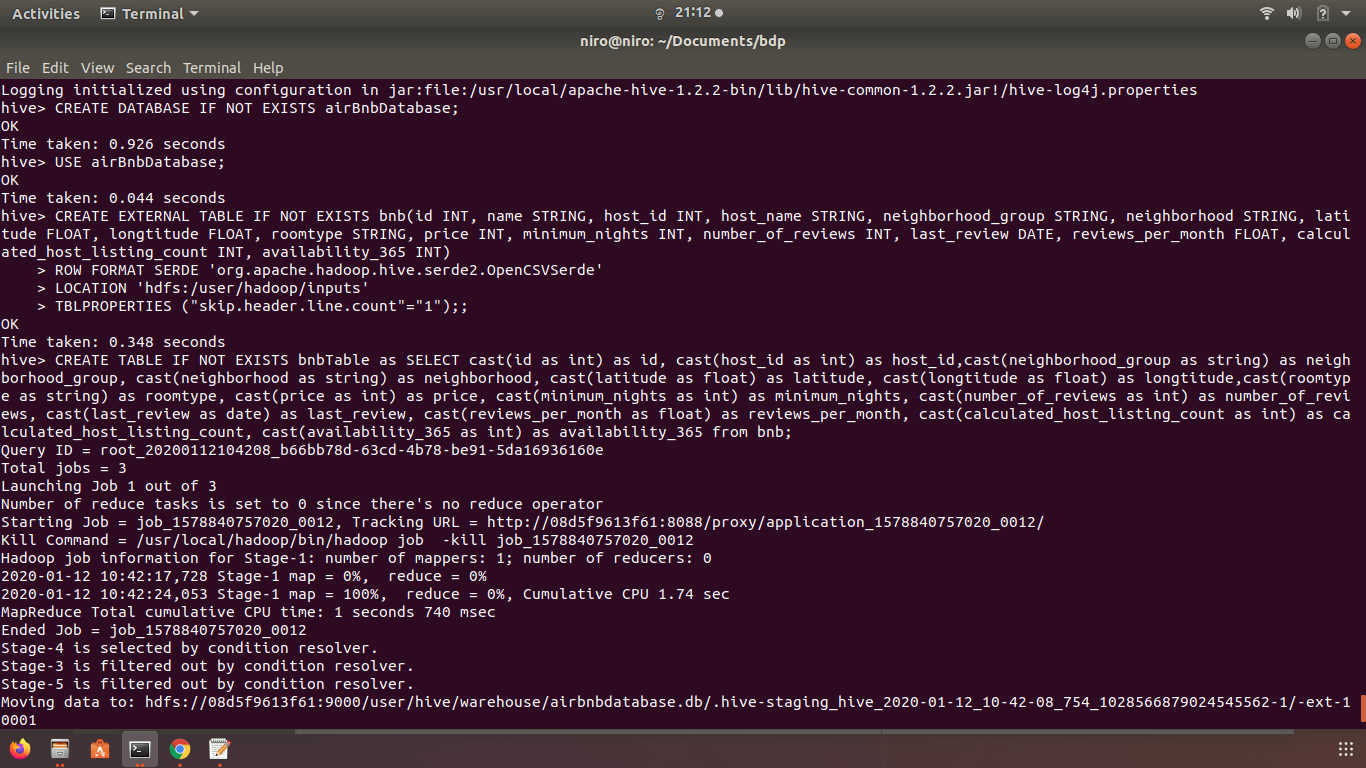
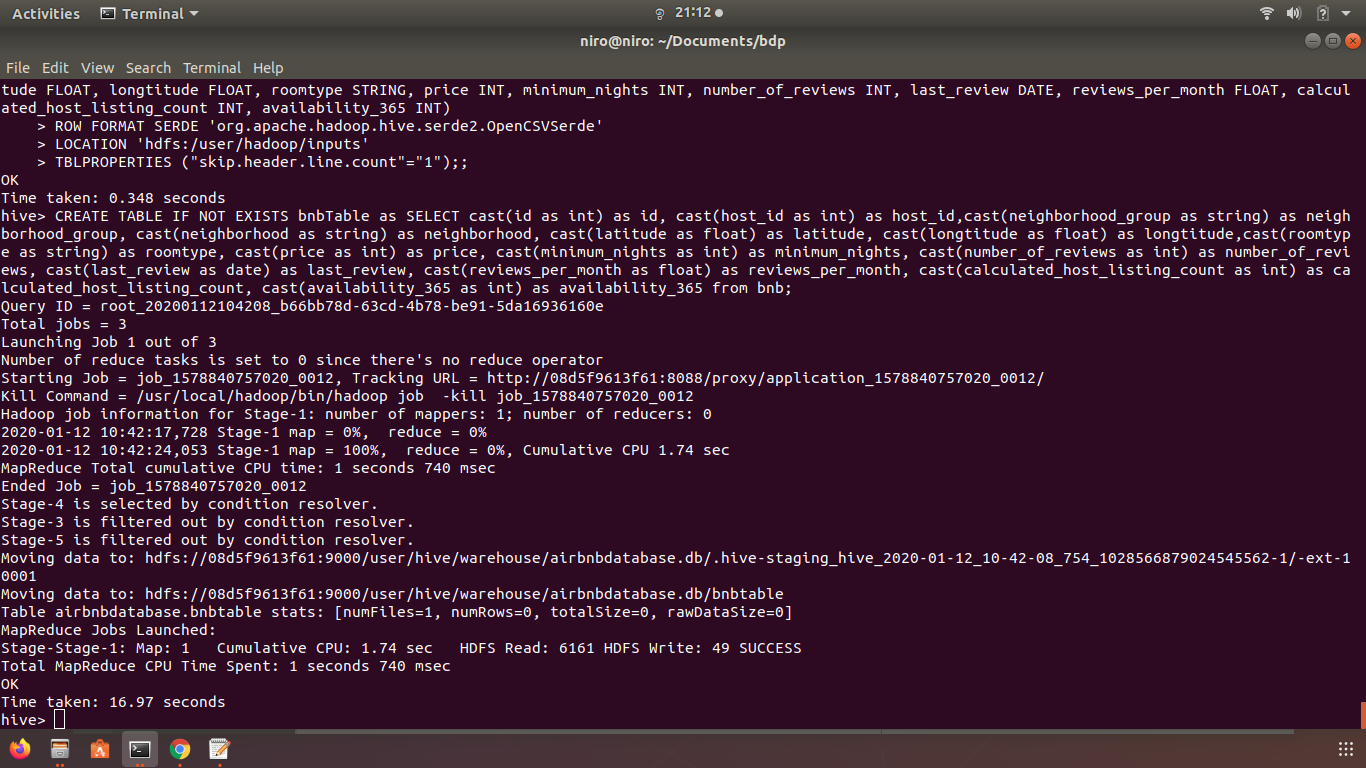
**Question 01 –** Count of 356 Availability Rentals

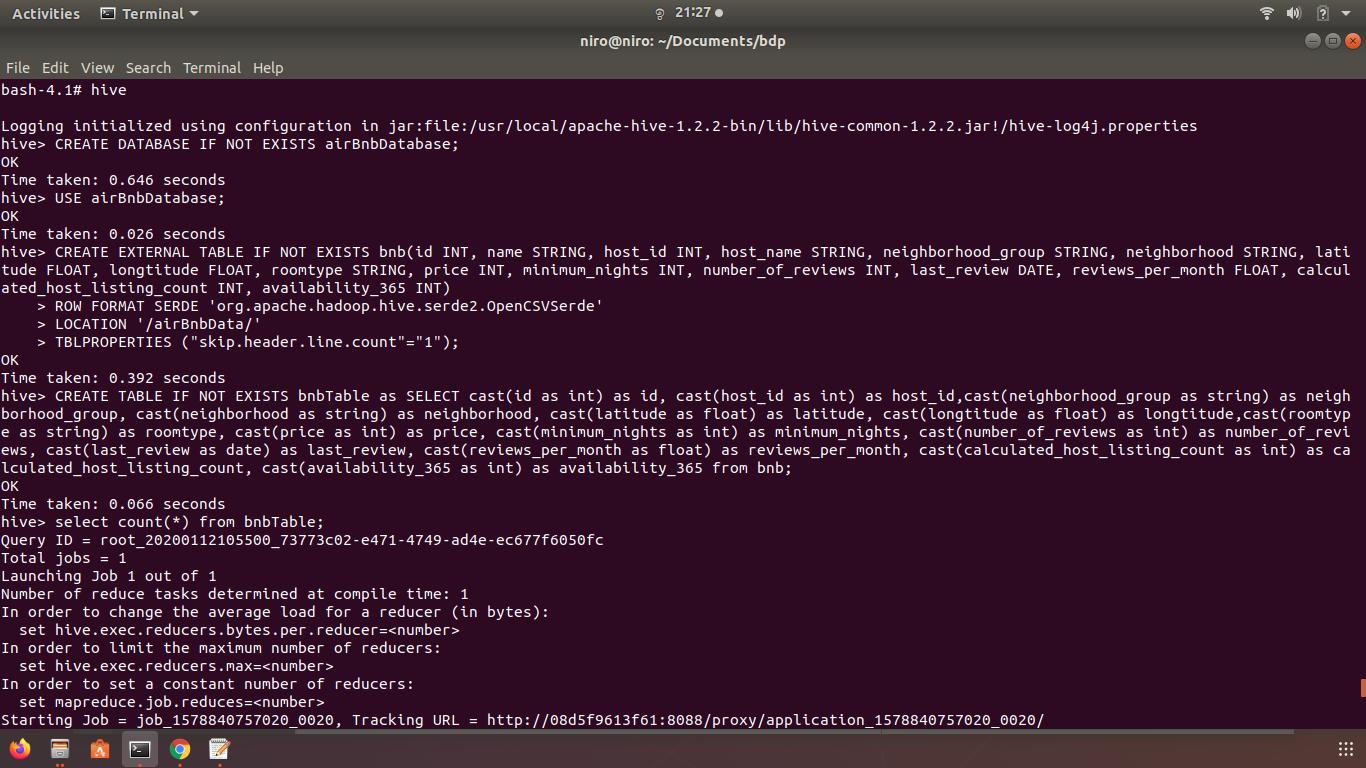
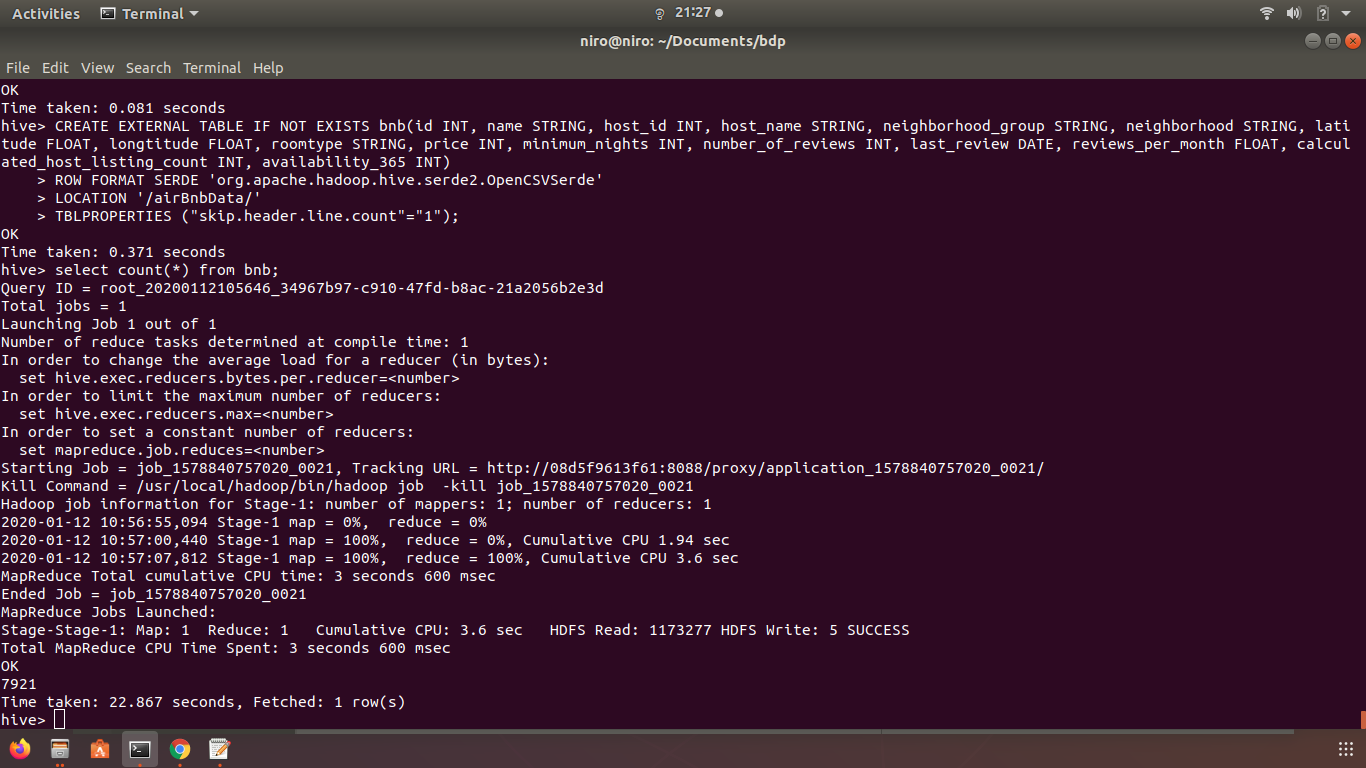
**Question 01:** output

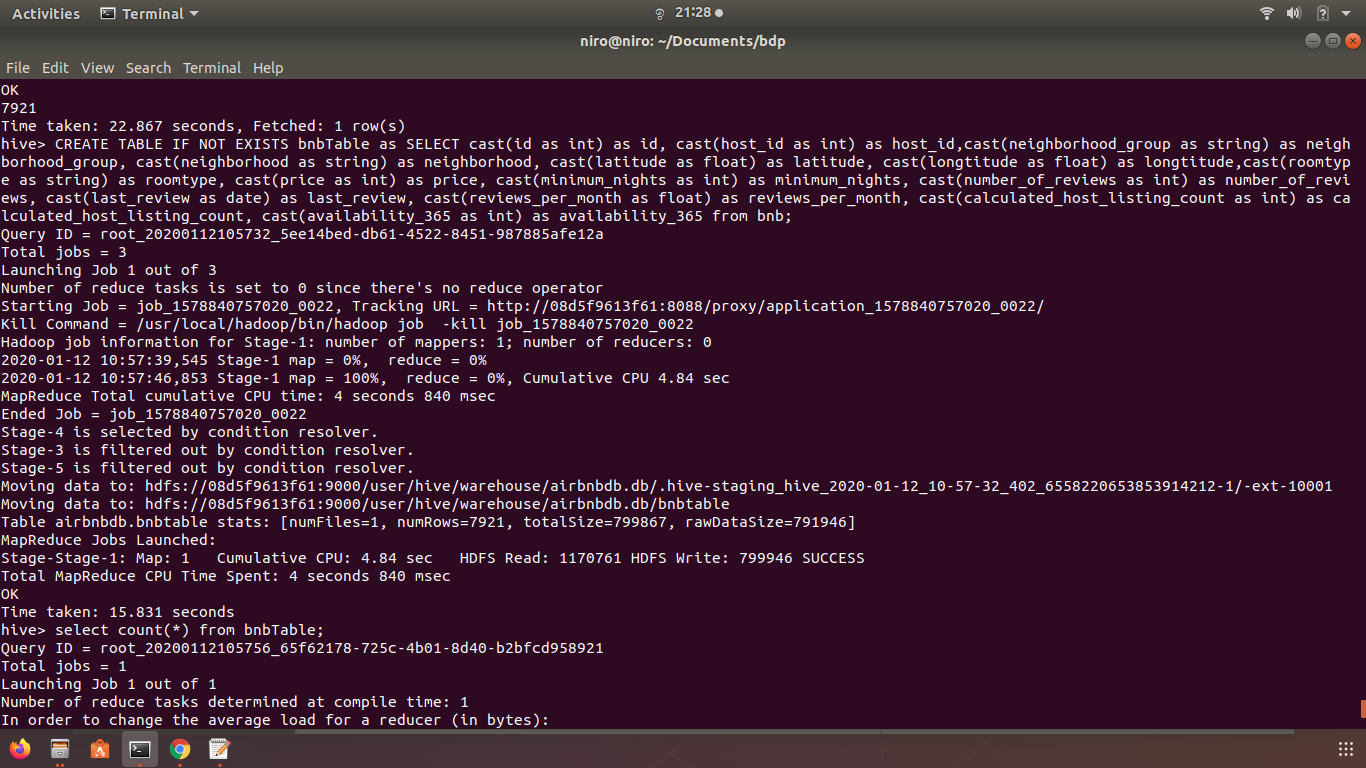
**Question 02** – Group Rentals by Neighborhood Group

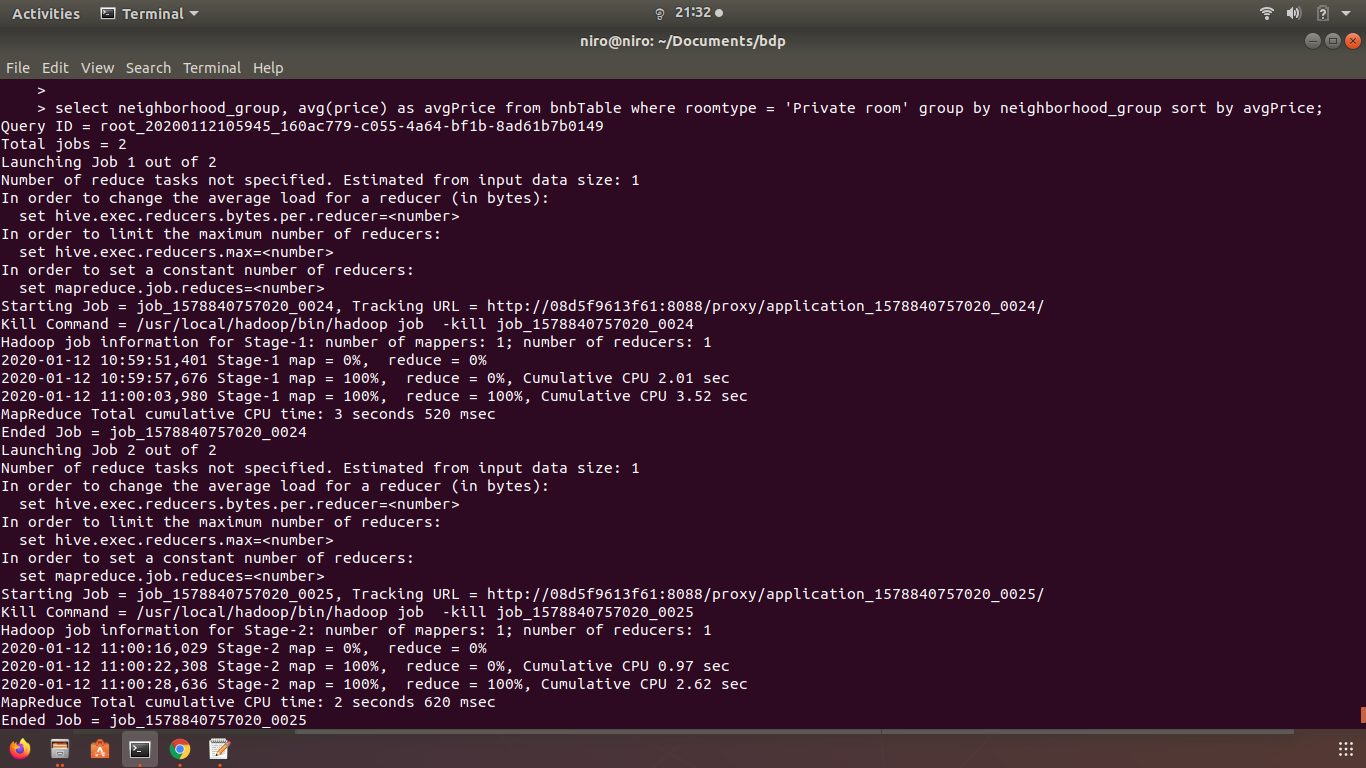
**Question 02:** output

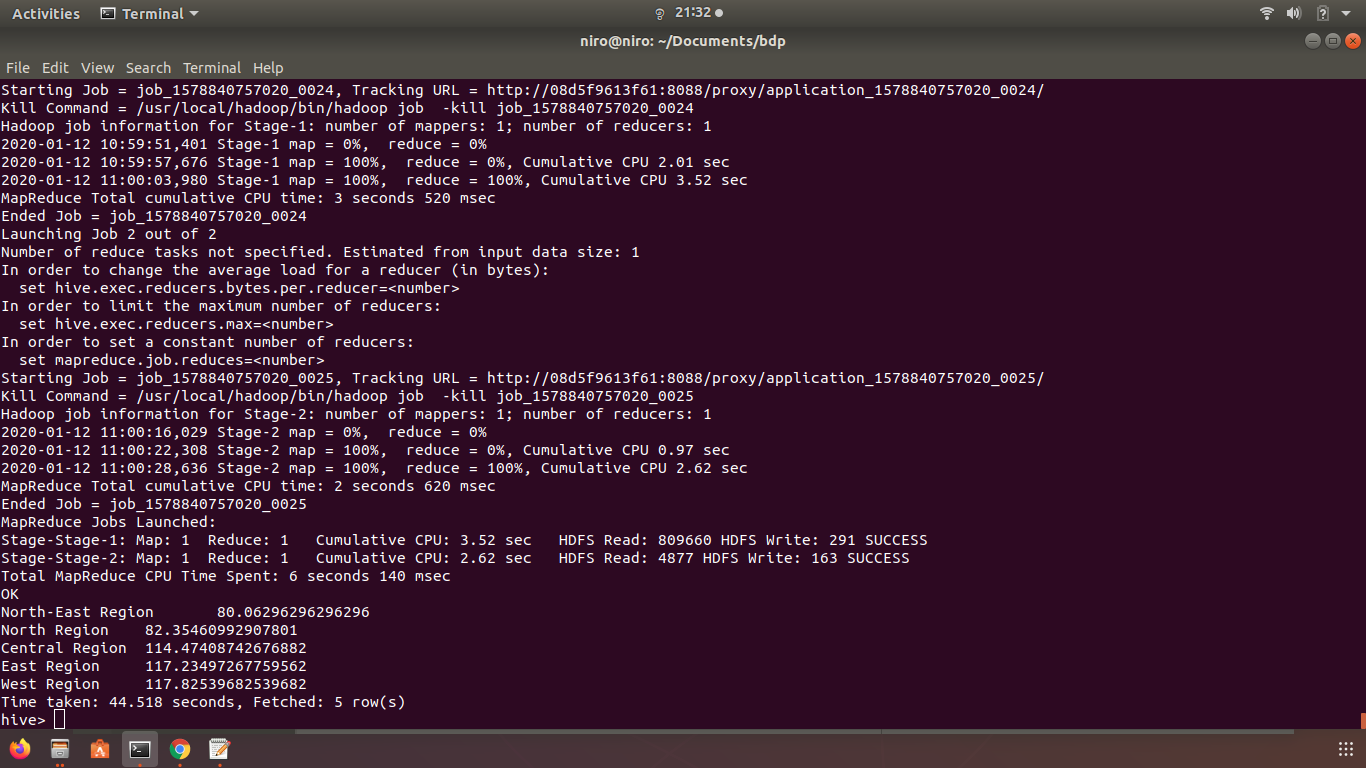
**2.2. Hive**

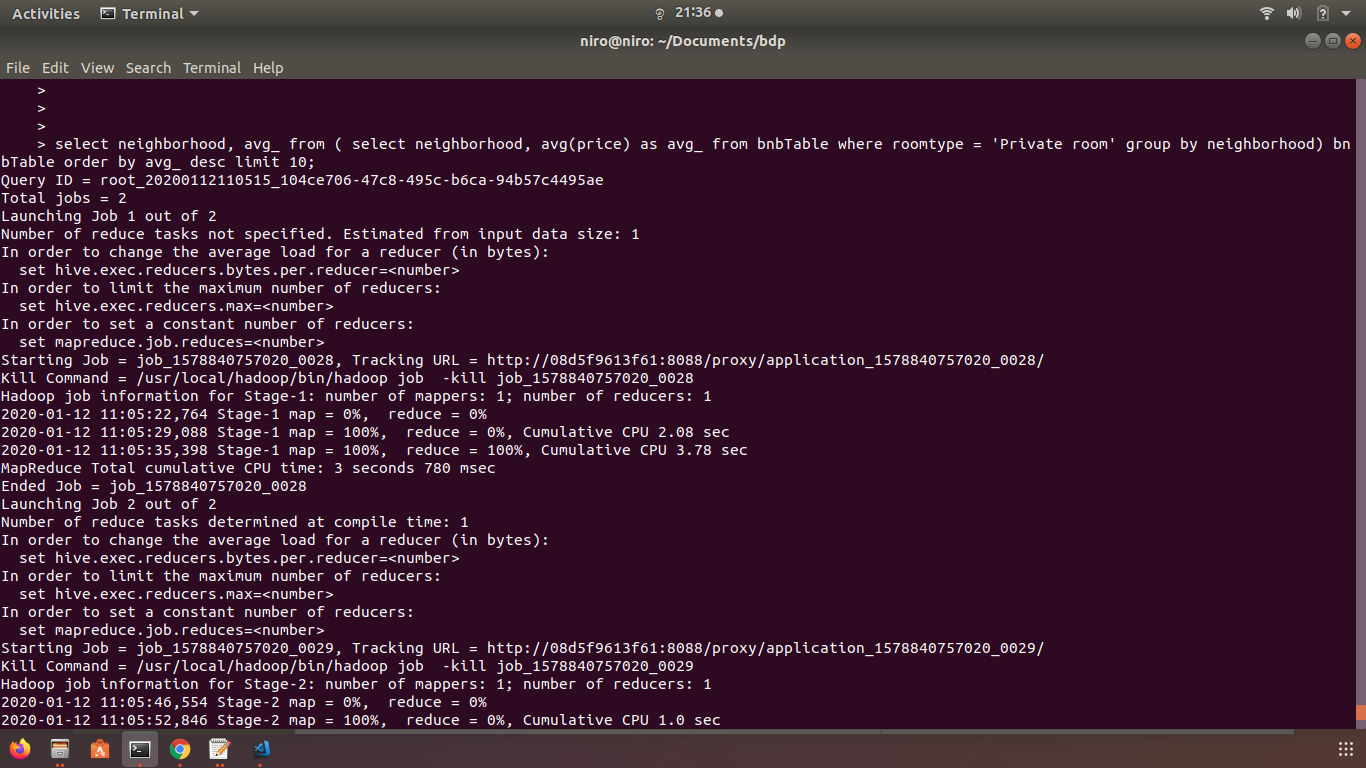
Setup: Create databases, create tables, insert and convert data

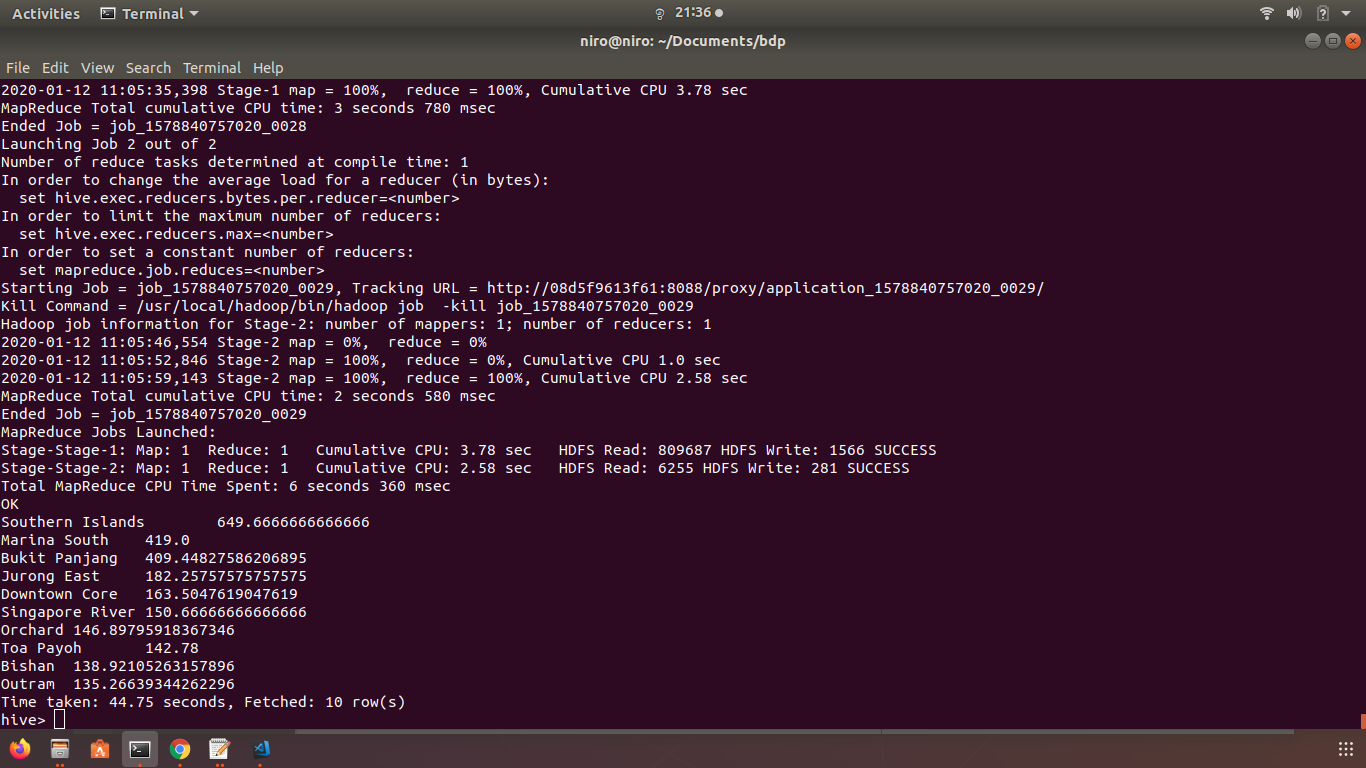




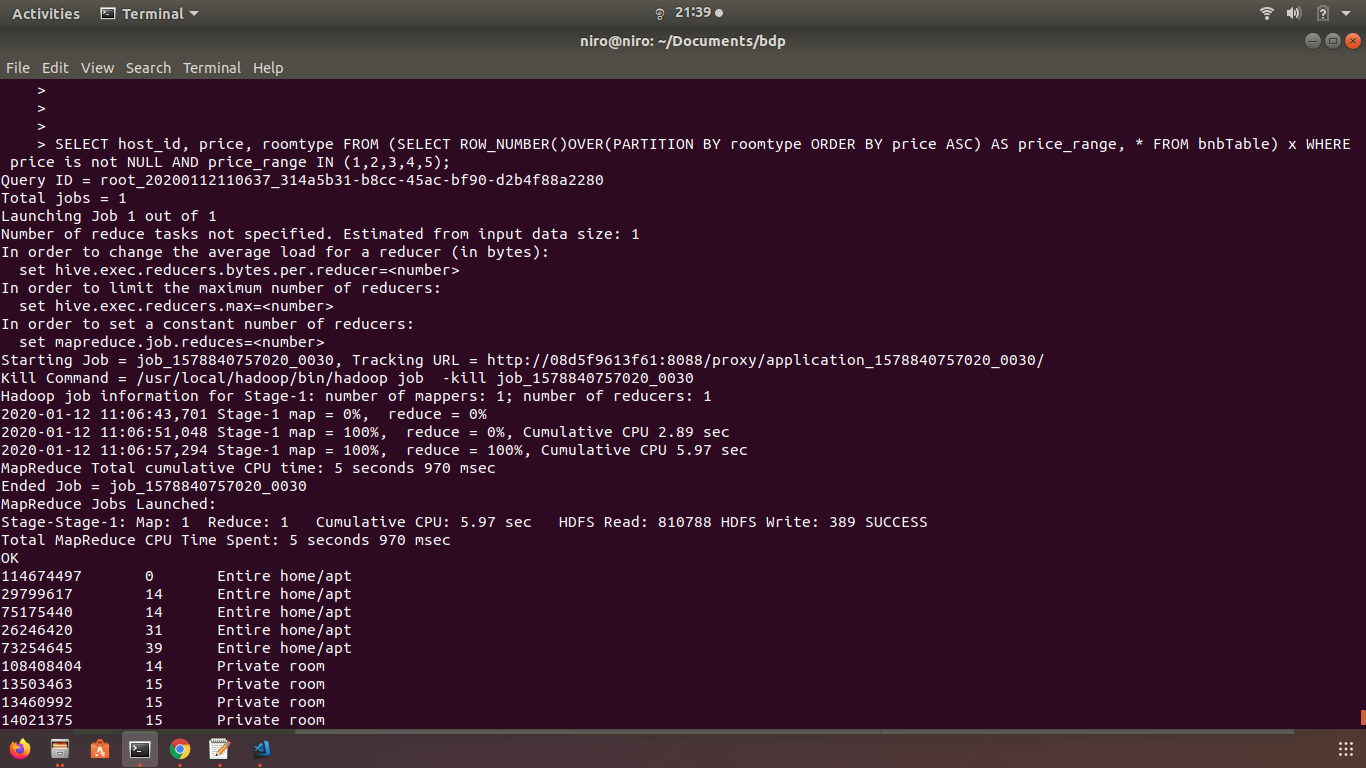
**Question 01** - Average price of Private room rental by neighborhood group

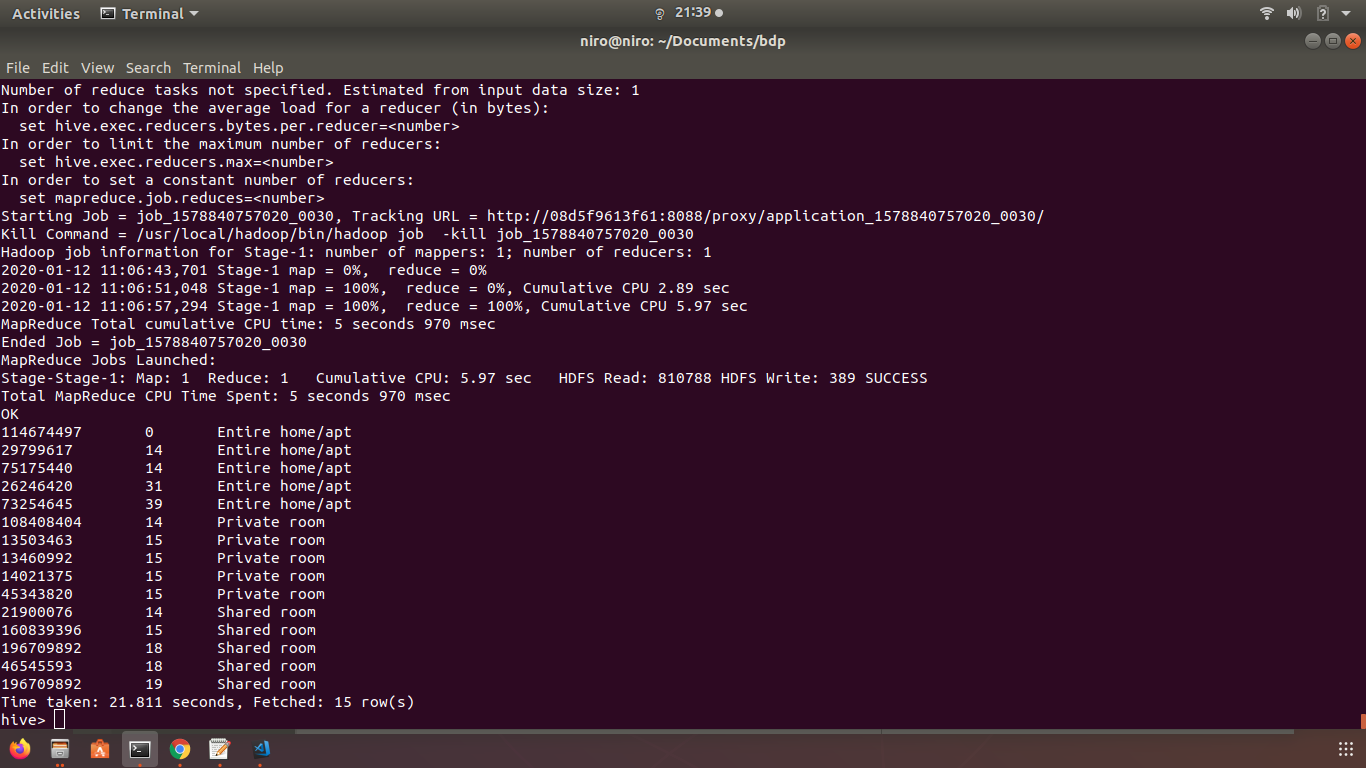
**Question 01:** output

**Question 02 -** Top 10 neighborhood based on Average price of Private room

**Question 02:** output

**Question 3** - The 5 lowest price properties per each Room Type

****

**Question 03:** output

**2.2. Spark**

**Question 1** - Percentage of owners who rent more than one property

// insert the data from csv

import org.apache.spark.sql.functions.{col, to\_date}

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

var upDF=spark.read

          .option("header", "true")

          .option("treatEmptyValuesAsNulls", "true")

          .option("mode","DROPMALFORMED")

          .option("delimiter", ",")

          .option("inferSchema", "true")

          .csv("/FileStore/tables/listings.csv")

//convert the string format to date in date columns

val df = upDF.columns.filter(colName =>colName.endsWith("\_review"))

.foldLeft(upDF) { (outputDF, columnName) =>

outputDF.withColumn(columnName, to\_date(col(columnName), "MM/dd/yyyy").cast("date"))

}

//write data into data frame

var rentalDf = df.toDF();

//drop null values

val totalHostsIds = rentalDf.filter(rentalDf("host\_id").isNotNull).select("host\_id").distinct().count()

//hosts that contains more than one rentals

rentalDf = rentalDf.where("calculated\_host\_listings\_count >1").select("host\_id").distinct()

//Count the number of hosts that contains more than one rentals

rentalDf = rentalDf.groupBy("host\_id").count().agg(count("host\_id").alias("count"))

// Convert the hosts numbers by 100 to produce percentage

val udf\_host\_percentage = udf((x:Int)=>{(x\*100)/totalHostsIds.toDouble})

//show results by adding new value named percentage

rentalDf.withColumn("percentage",udf\_host\_percentage(rentalDf("count"))).show()

**Question 01:** output

**Question 02:** Histogram of number of rentals reviewed over time (based on last review) in mouth granularity.

// insert the data from csv

import org.apache.spark.sql.functions.{col, to\_date}

var upDF=spark.read

          .option("header", "true")

          .option("treatEmptyValuesAsNulls", "true")

          .option("mode","DROPMALFORMED")

          .option("delimiter", ",")

          .option("inferSchema", "true")

          .csv("/FileStore/tables/listings.csv")

// convert the string format to date in date columns

val df = upDF.columns.filter(colName =>colName.endsWith("\_review"))

.foldLeft(upDF) { (outputDF, columnName) =>

outputDF.withColumn(columnName, to\_date(col(columnName), "MM/dd/yyyy").cast("date"))

}

df.count

//write data into data frame

var rentalDf = df.toDF();

//drop null values

rentalDf = rentalDf.select("last\_review").where("last\_review IS NOT NULL")

//get year with month substring "yyyy-MM"

val udf\_get\_month = udf((x:String)=>x.slice(0,7))

//add new column called last\_review\_month

rentalDf = rentalDf.withColumn("last\_review\_month",udf\_get\_month(rentalDf("last\_review")))

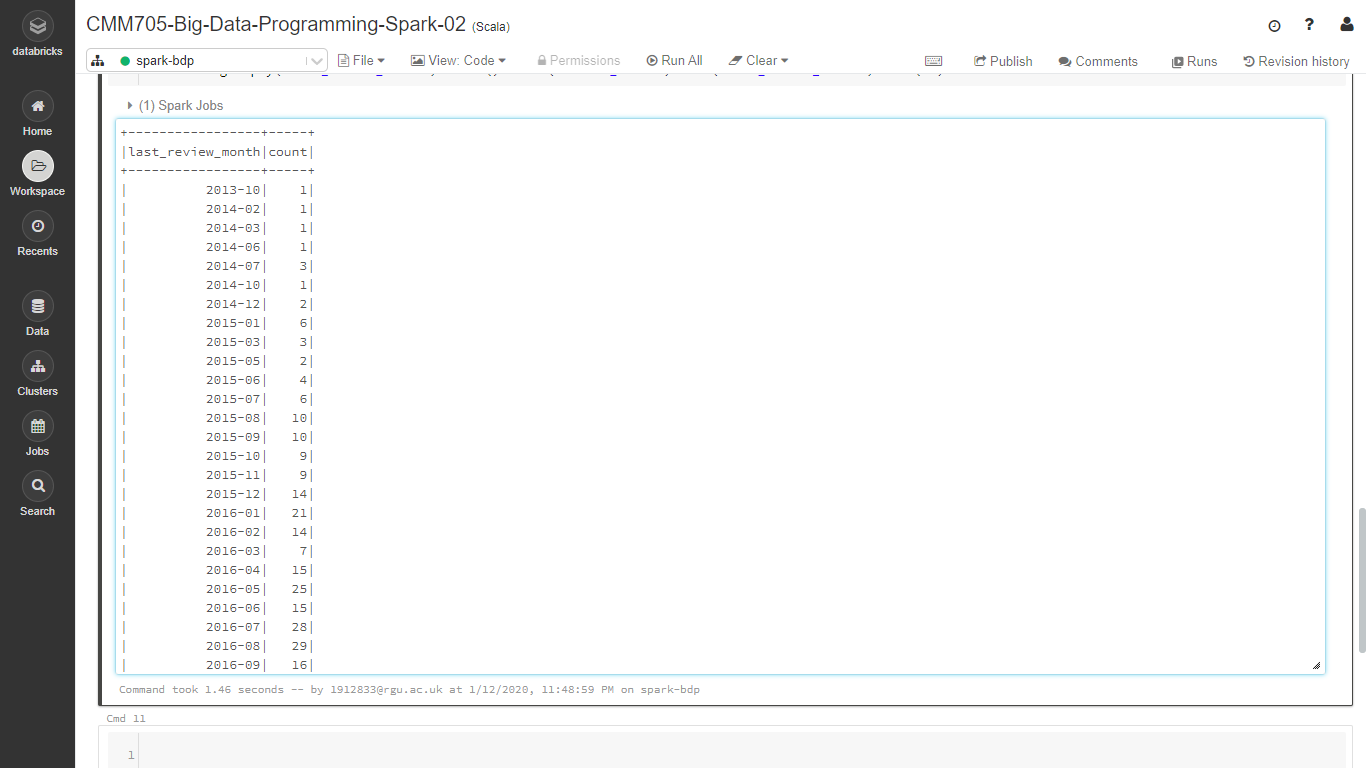
//group last review

//display count per month granularity

rentalDf.groupBy("last\_review\_month").count().alias("review\_count").sort("last\_review\_month").count

//group last review and show count per month granularity

rentalDf.groupBy("last\_review\_month").count().alias("review\_count").sort("last\_review\_month").show(61)

**Question 02:** output

**Question 03** - Number of rentals that are available all 365 days of the year for each neighborhood, that are in the neighborhood which have top 5 average rental prices.

// insert the data from csv

import org.apache.spark.sql.functions.{col, to\_date}

var upDF=spark.read

          .option("header", "true")

          .option("treatEmptyValuesAsNulls", "true")

          .option("mode","DROPMALFORMED")

          .option("delimiter", ",")

          .option("inferSchema", "true")

          .csv("/FileStore/tables/listings.csv")

//convert the string format to date in date columns

val df = upDF.columns.filter(colName =>colName.endsWith("\_review"))

.foldLeft(upDF) { (outputDF, columnName) =>

outputDF.withColumn(columnName, to\_date(col(columnName), "MM/dd/yyyy").cast("date"))

}

//write data into data frame

var rentalDf = df.toDF();

//filter dataframe that availability equals 365

rentalDf = rentalDf.where("availability\_365 = 365");

//create temporary view to store data

rentalDf.createOrReplaceTempView("Available356DaysView")

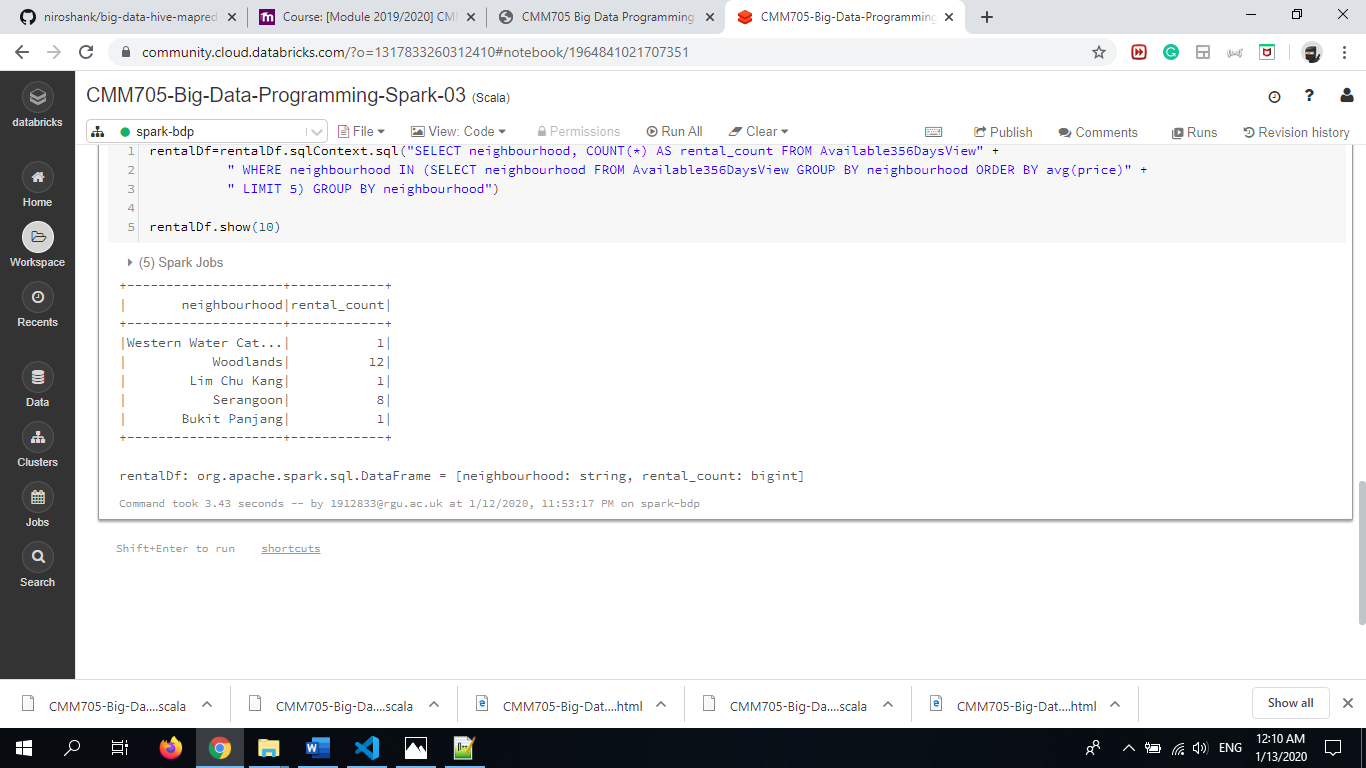
rentalDf=rentalDf.sqlContext.sql("SELECT neighbourhood, COUNT(\*) AS rental\_count FROM Available356DaysView" +

          " WHERE neighbourhood IN (SELECT neighbourhood FROM Available356DaysView GROUP BY neighbourhood ORDER BY avg(price)" +

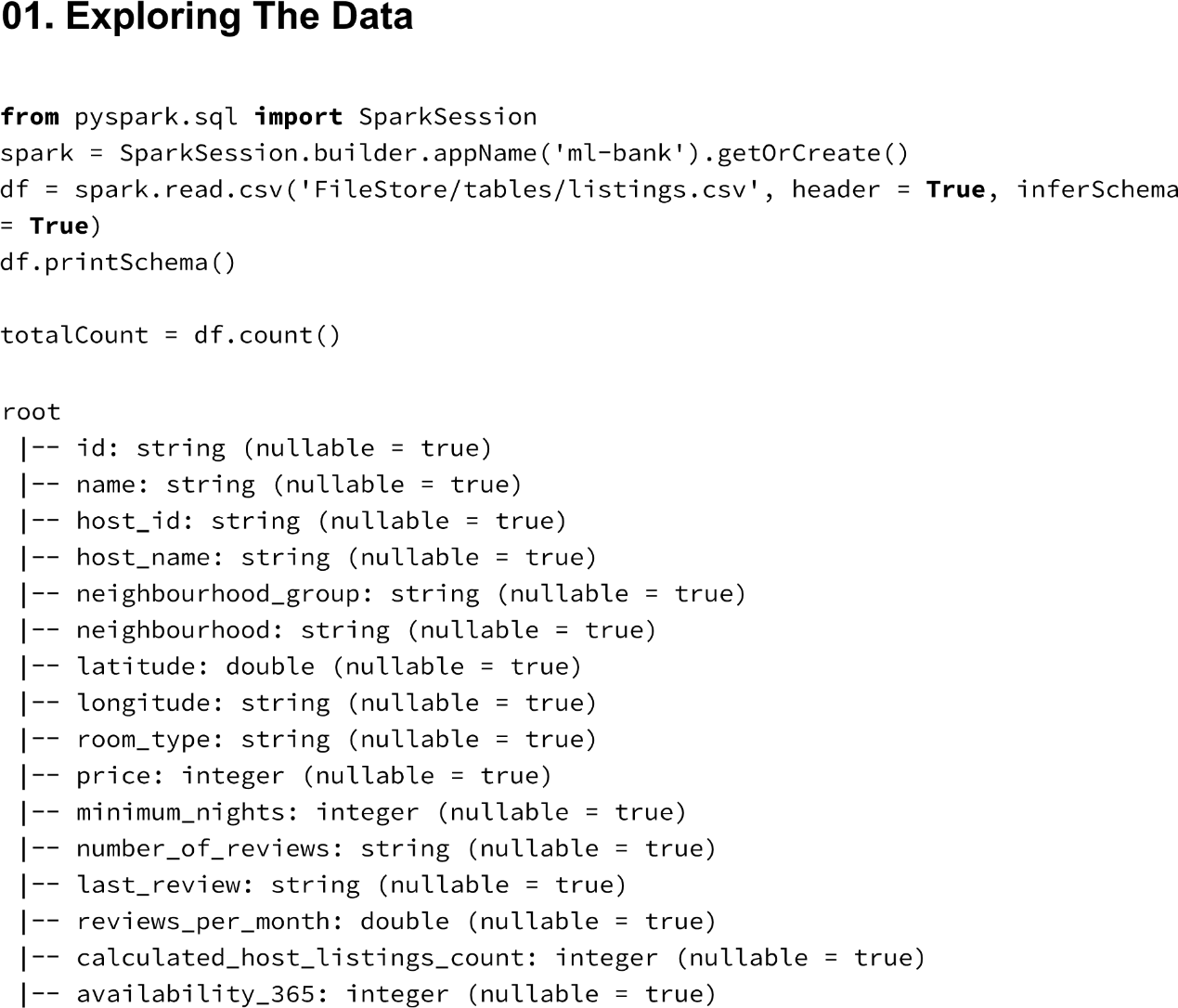
          " LIMIT 5) GROUP BY neighbourhood")

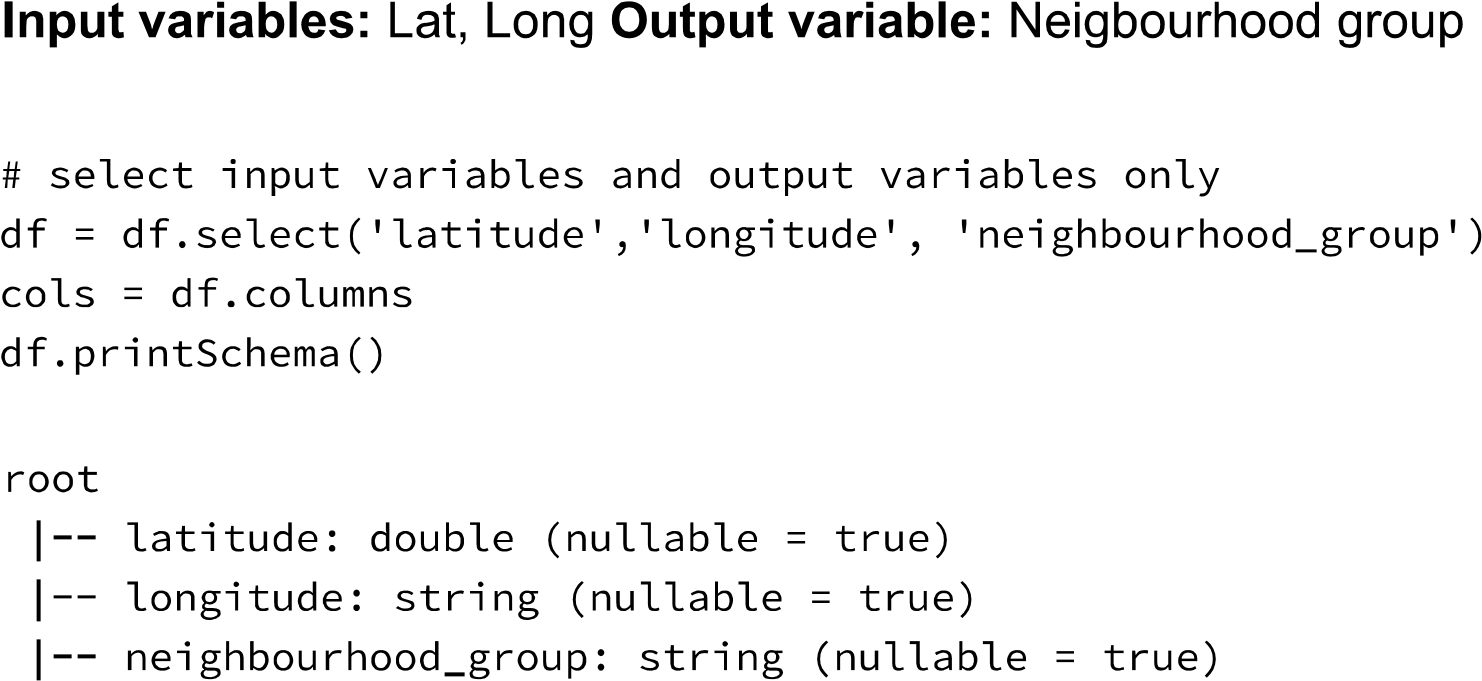
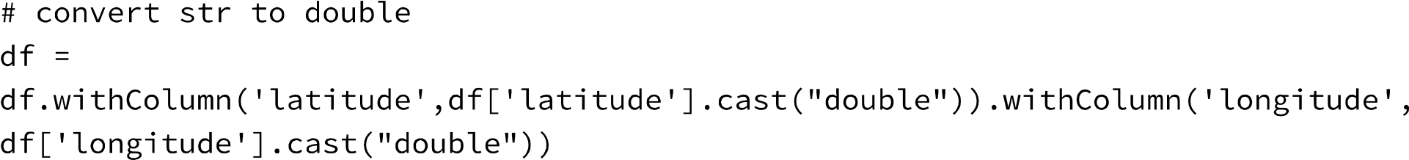
rentalDf.show(10)

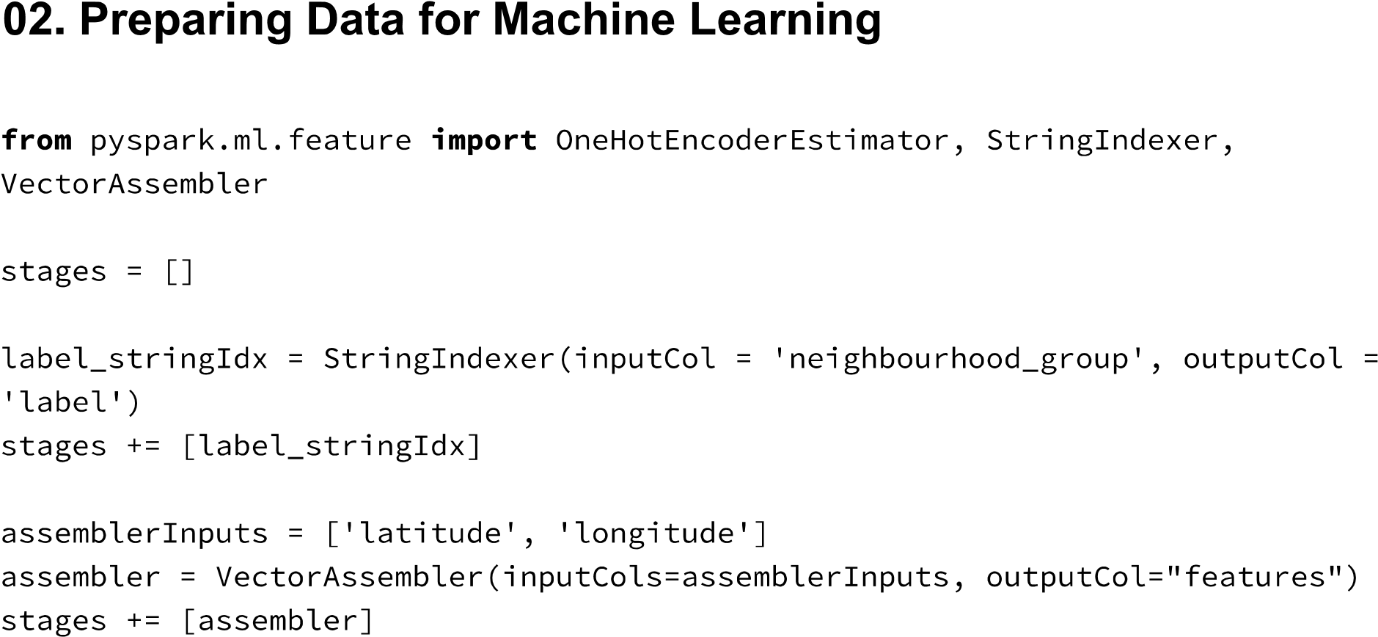
**Question 03:** output

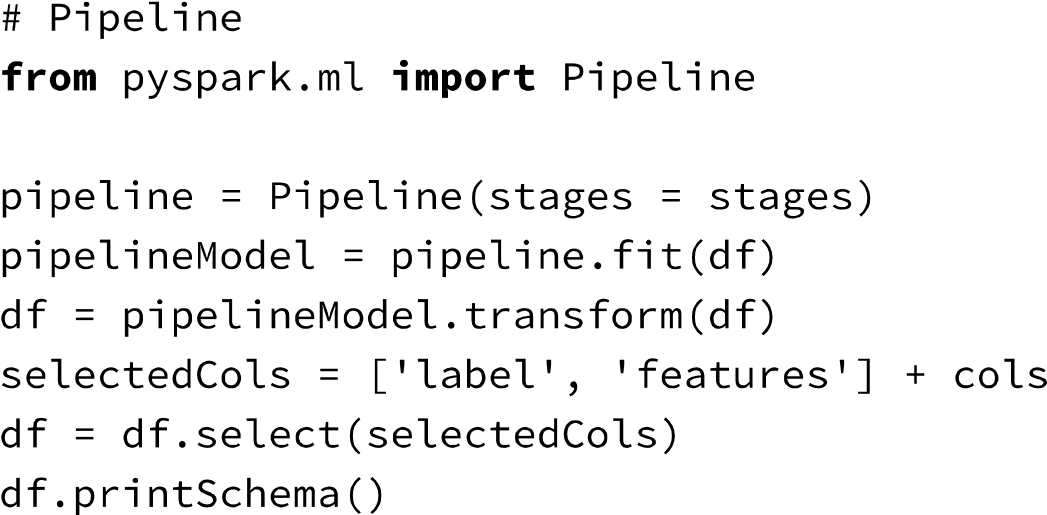


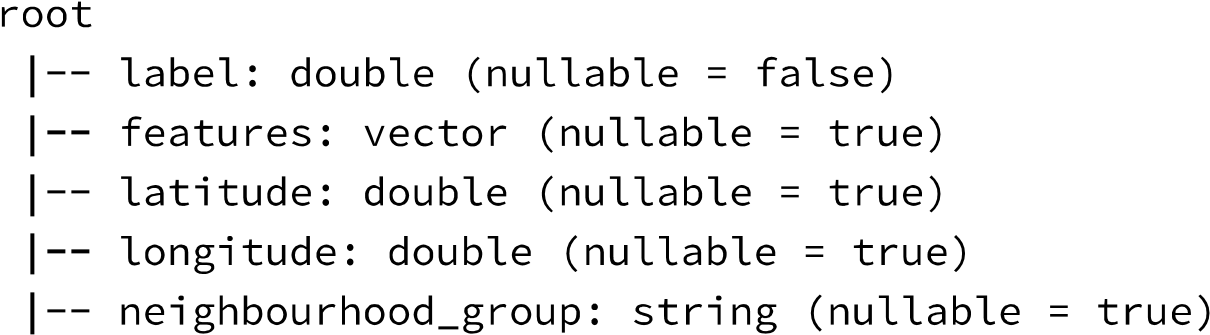
1. **Steps for training and validating the model**

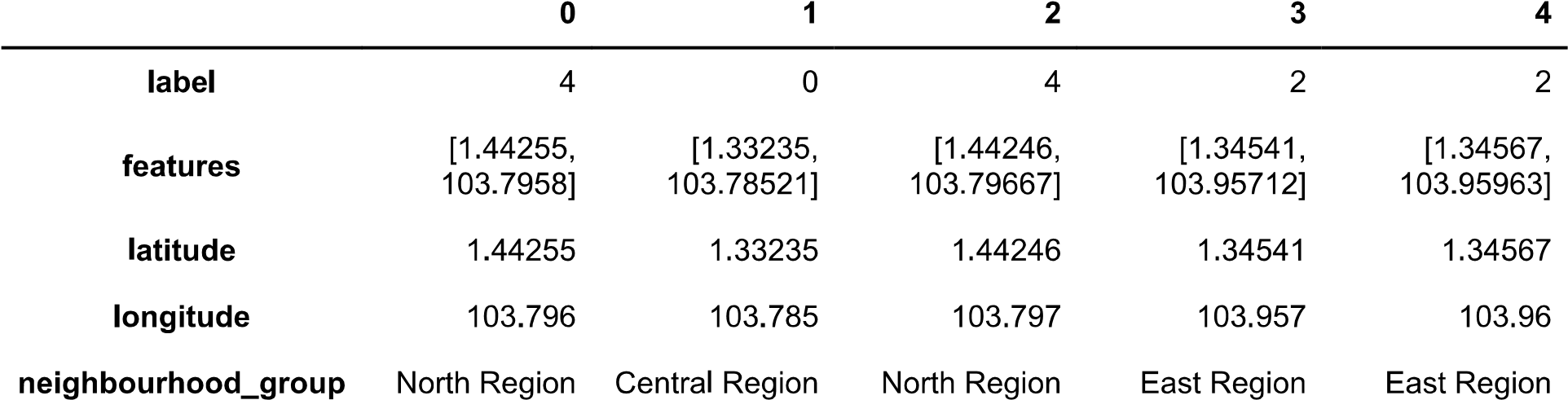


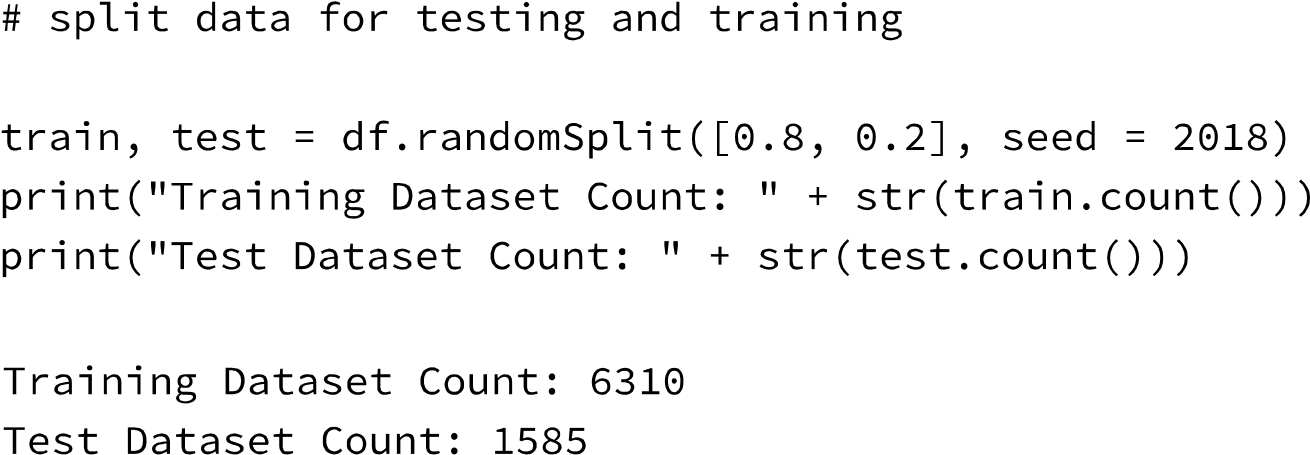


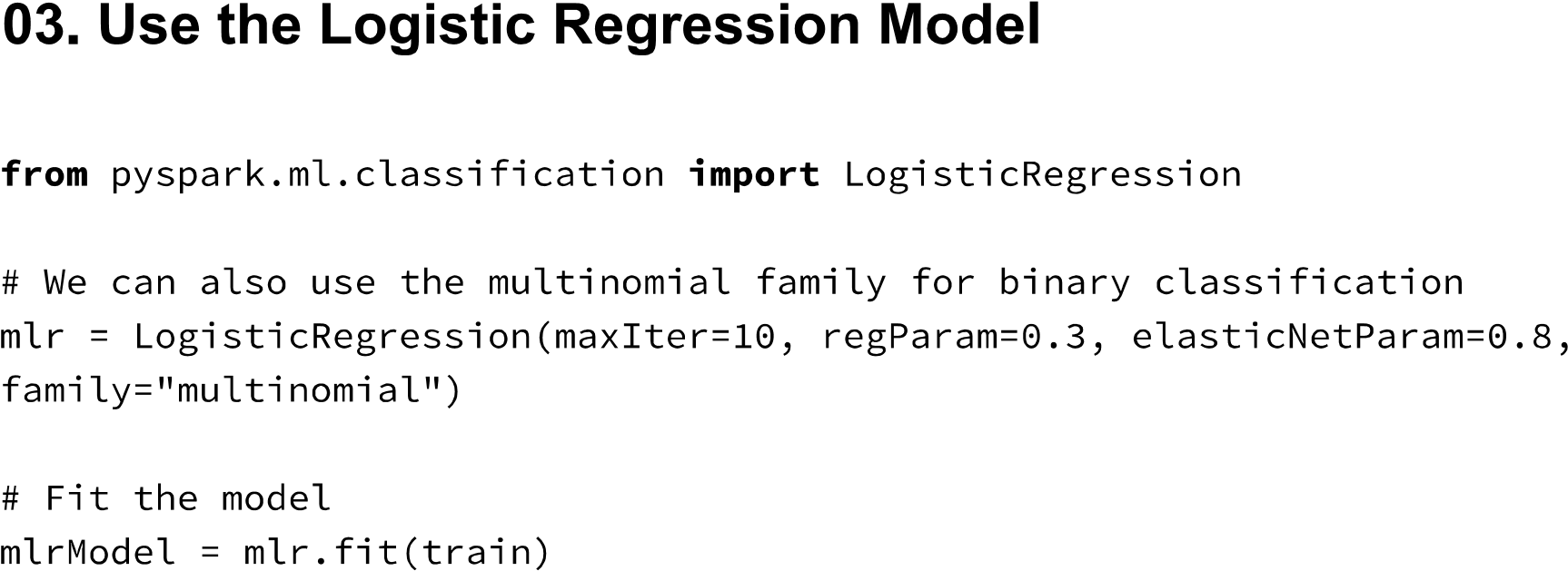


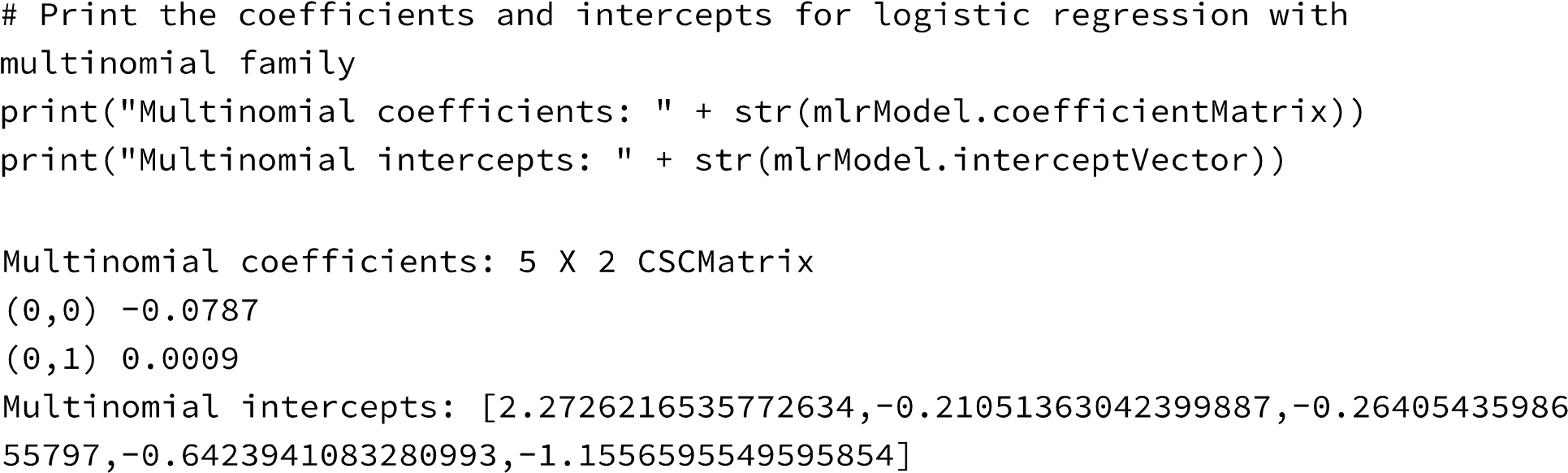


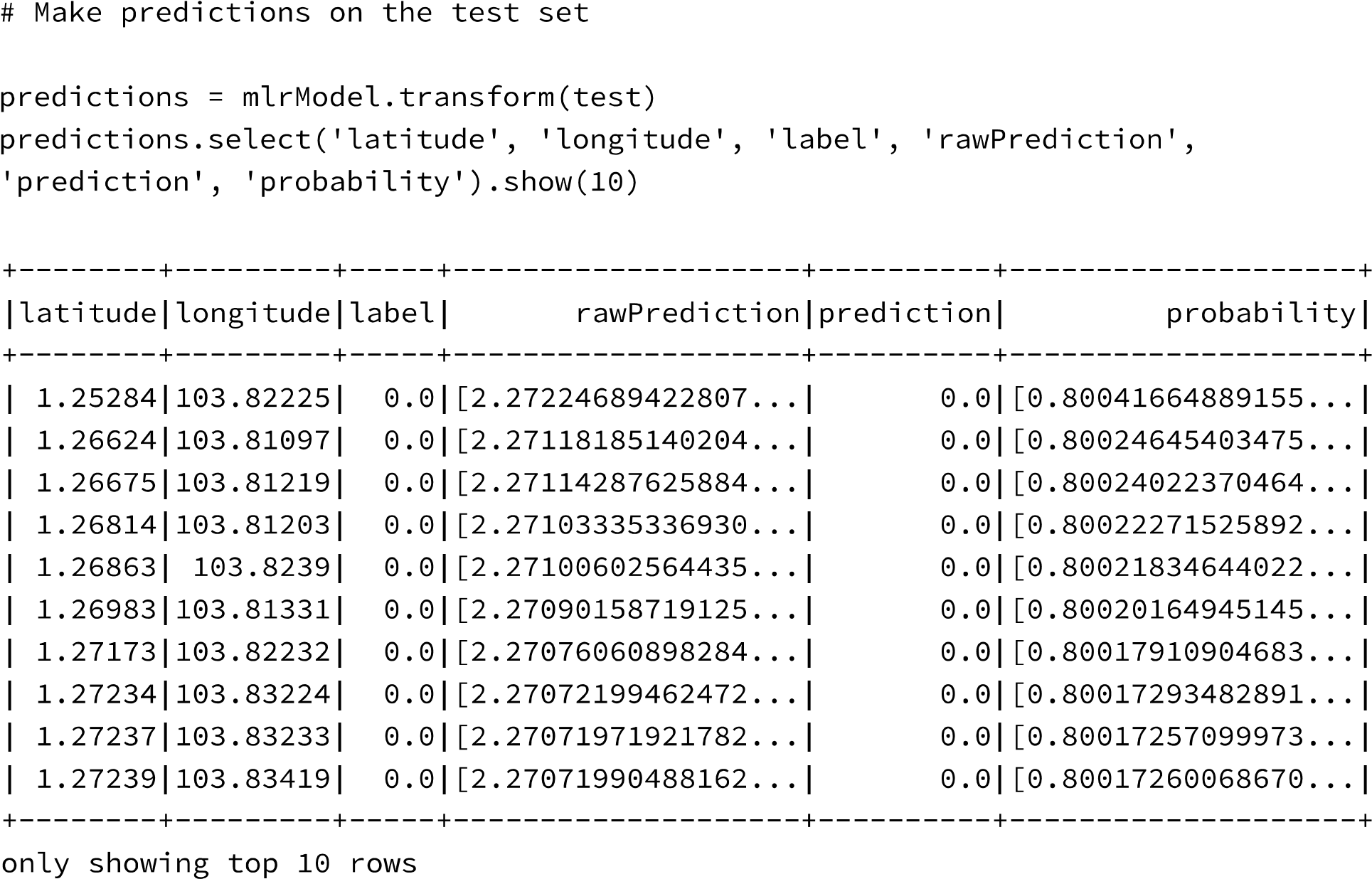


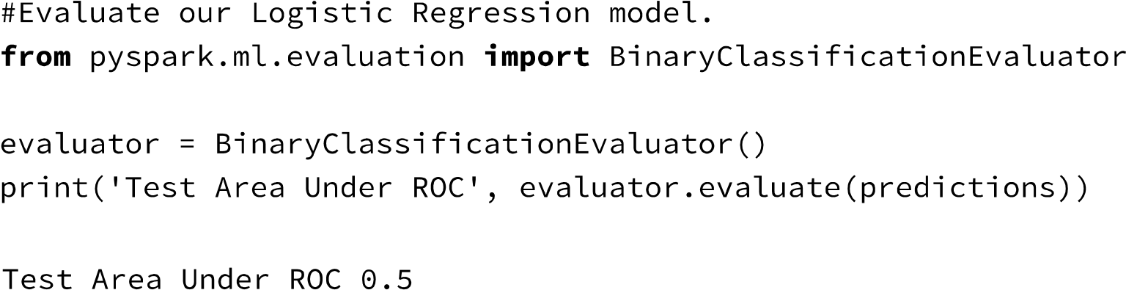


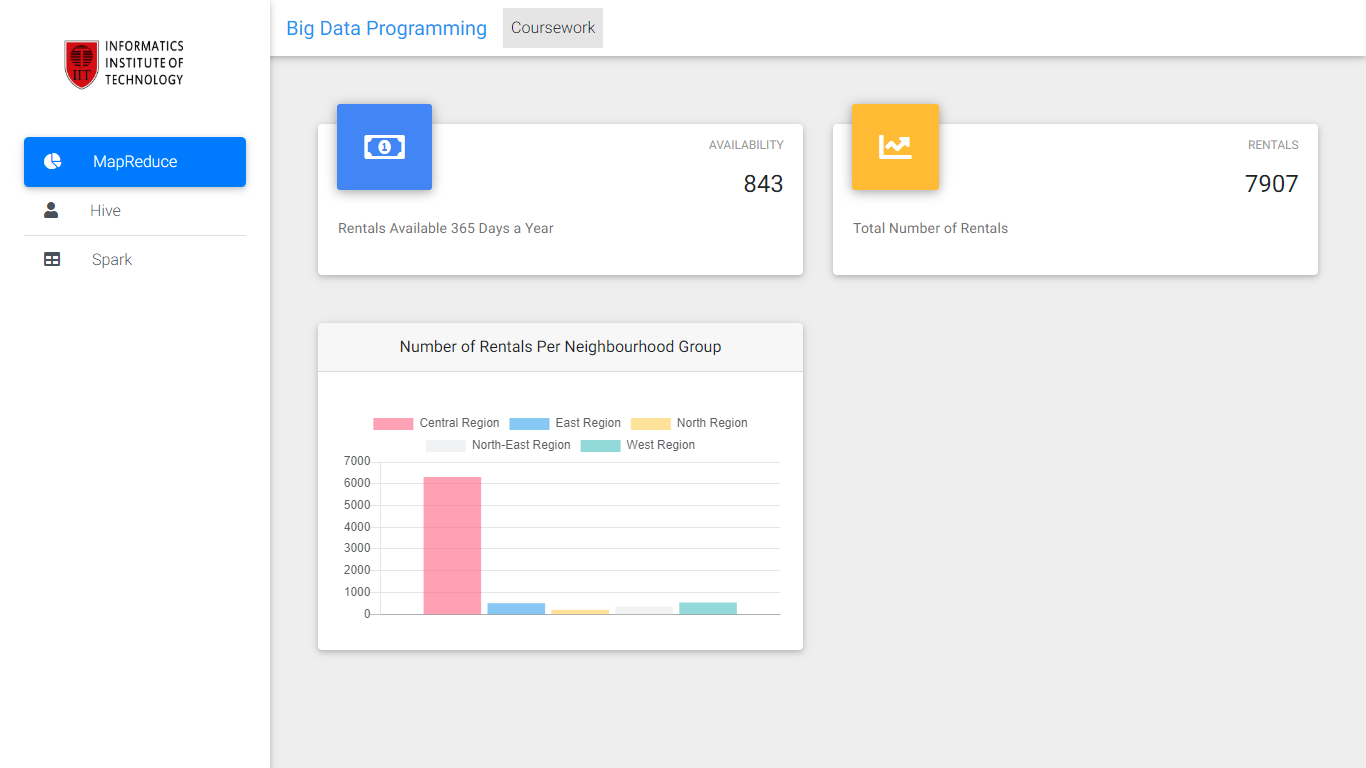


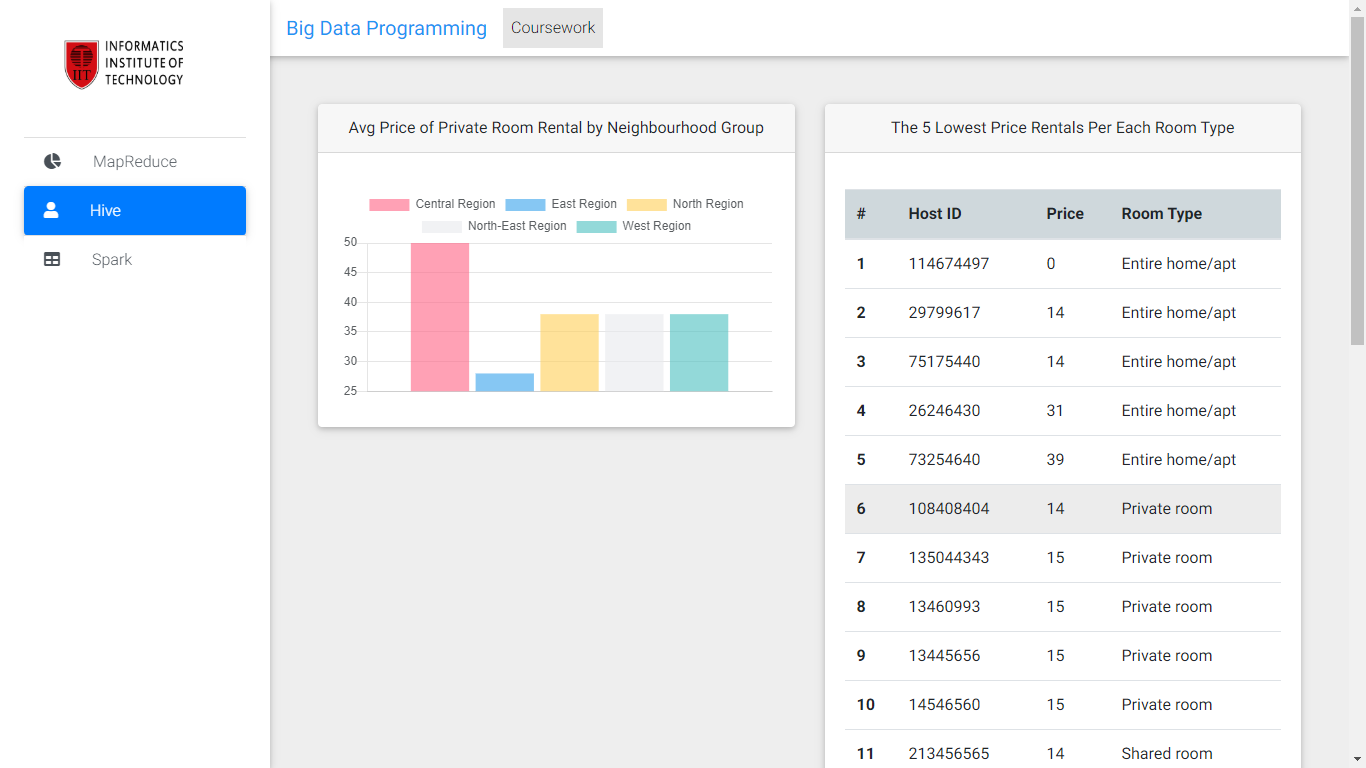


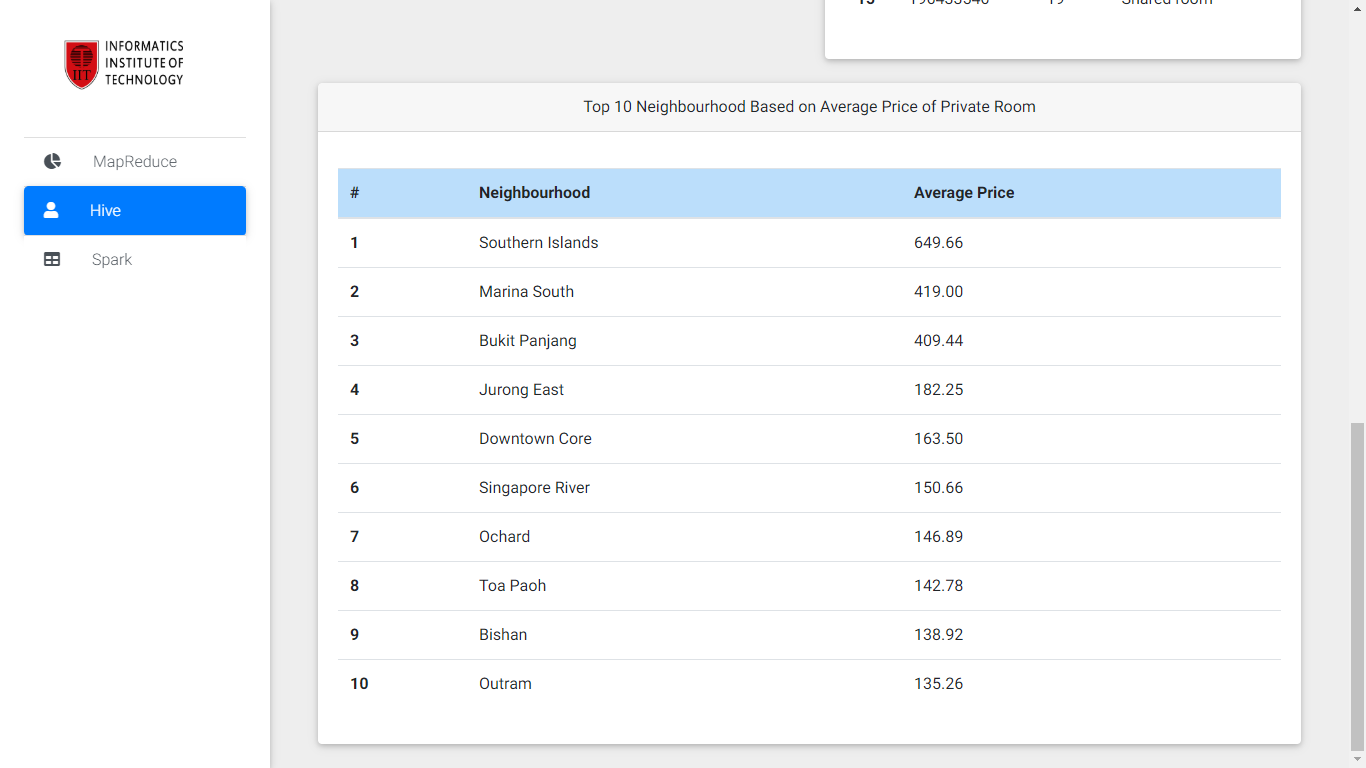


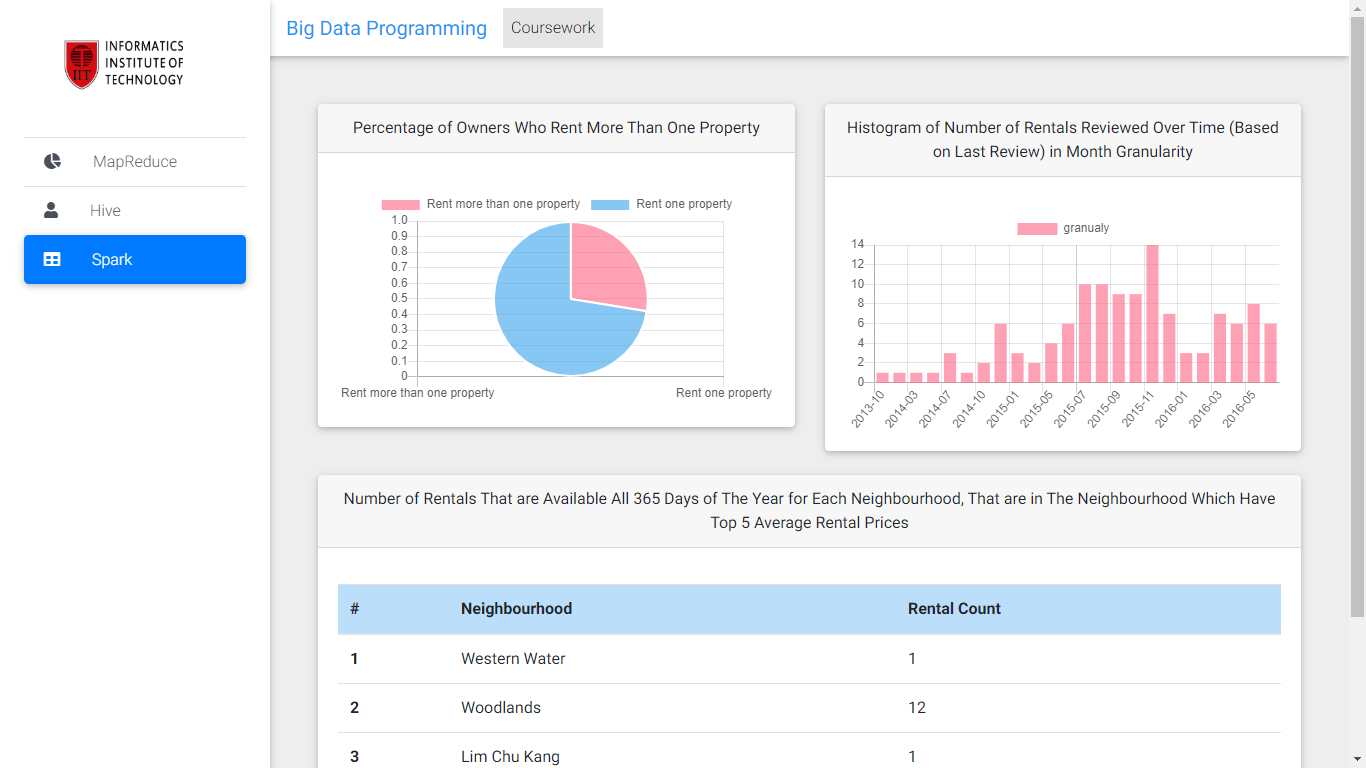


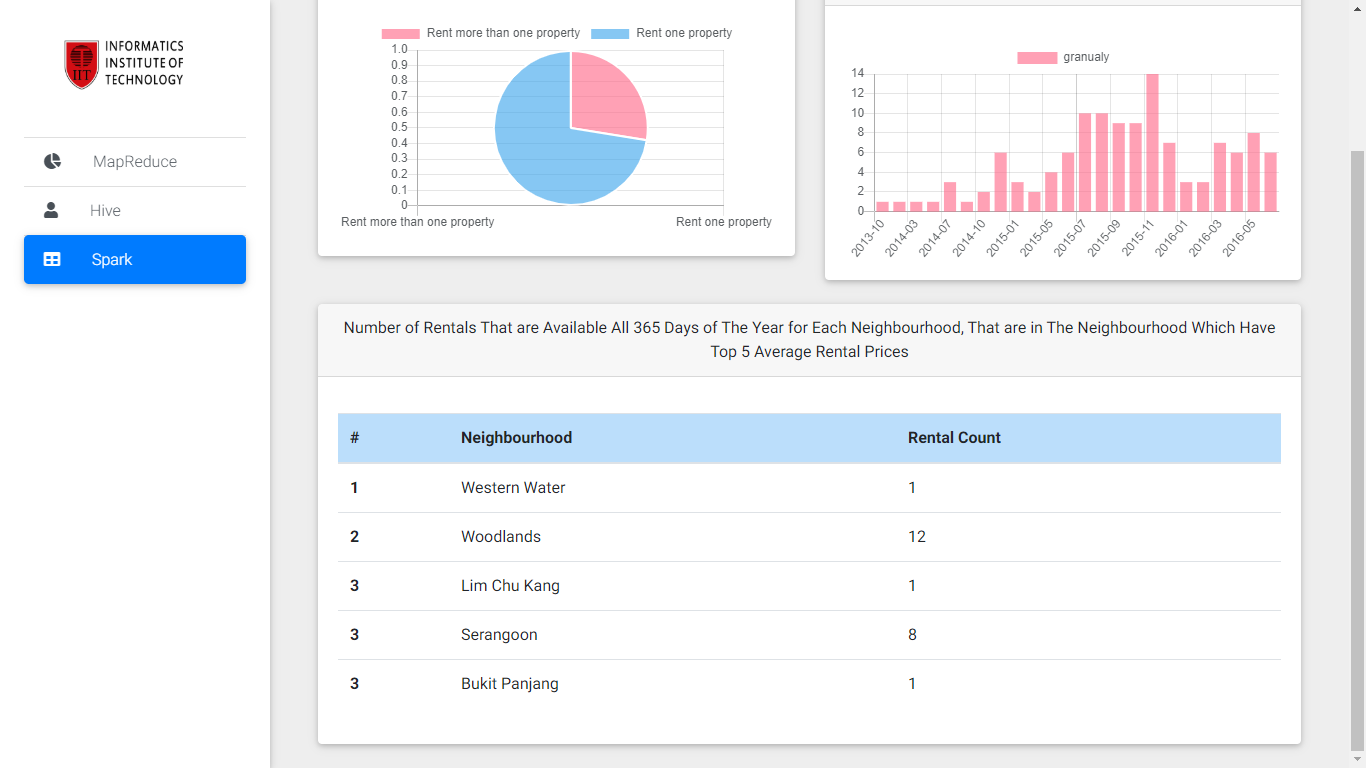


1. **Screenshots of the Dashboard**









**Github Repository:** <https://github.com/niroshank/big-data-hive-mapred-spark>