INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration With

ROBERT GORDON UNIVERSITY ABERDEEN

Big Data Programming Report

MSc. in Big Data Analytics

2019/2020

By

Dulan Manujaya

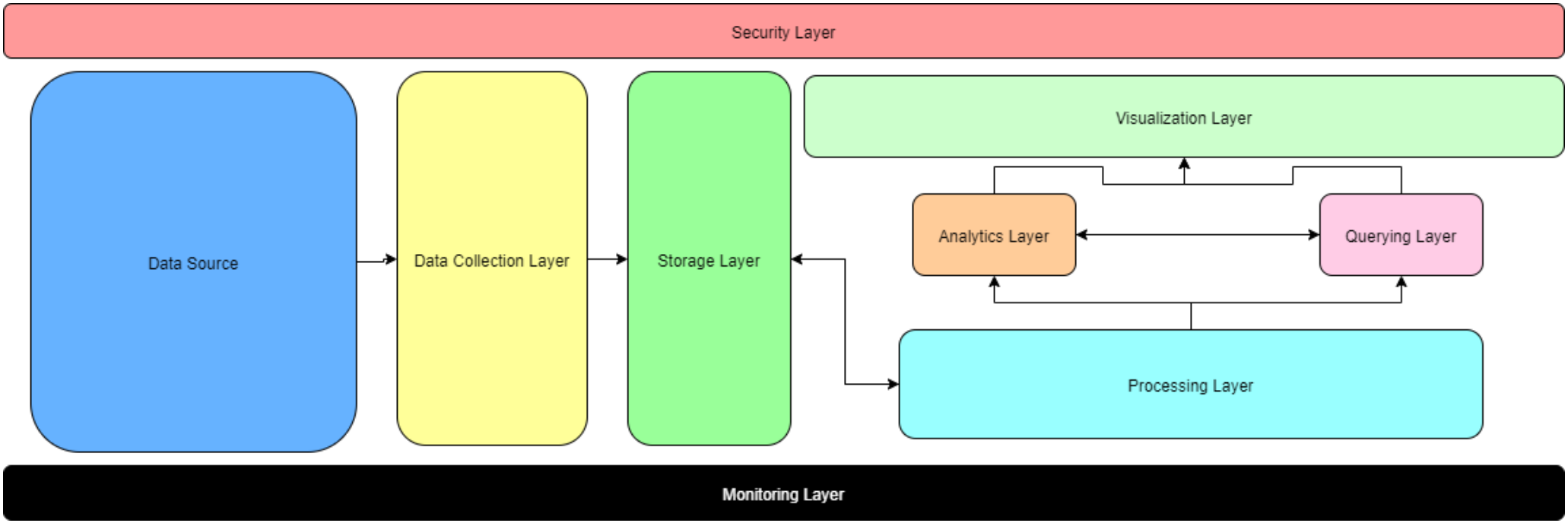
IIT No : **2019528** RGU Id : **1913768**

CMM705

Big Data Programming

Contents

Section 1



Data Source: - Airbnb Website

Data Collection Layer: - ETL

Storage Layer: - HDFS, Cloud (S3), NoSQL

Processing Layer: - Batch Processing

Querying Layer: - Hive, Pig, SparkSQL, MapReduce

Analytics Layer: - Machine Learning, Predictive Modelling, Statistical Analytics

Visualization Layer: - JavaScript Webpage

Section 2.1

2.1.1

```
public class RecordCountDriver
        extends Configured implements Tool{
    private  static  class RecordMapper extends Mapper<LongWritable, Text, Text, LongWritable>{
        public void map(LongWritable lineOffset, Text record, Context output)
        throws IOException, InterruptedException {
            String row = record.toString();
            String clean_row = row.replaceAll("[\\n\\t ]", "");
            String[] tokens = clean_row.split(",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)");

            int length = tokens.length;
            String col15 = tokens[length - 1];


            if (!col15.contains("availability_365")) {


                int colInt15 = Integer.parseInt(col15);


                if (colInt15 == 365) {
                    // context.write(new IntWritable(colInt3), new IntWritable(colInt0));
                    output.write(new Text("Count"),new LongWritable(1));
                }
                else{
                    return;
                }
            }
            else{
                return;
            }

        }
    }
```

```java
public int run(String[] args) throws Exception {
    // TODO Auto-generated method stub
    @SuppressWarnings("deprecation")
    Job job = new Job(getConf(), "Record Count");
    job.setJarByClass(getClass());

    job.setMapperClass(RecordMapper.class);
    job.setReducerClass(LongSumReducer.class);

    job.setNumReduceTasks(1);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(LongWritable.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(LongWritable.class);



    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    return job.waitForCompletion(true) ? 0 : 1;
}

public static void main(String[] args) throws Exception {
    int jobStatus = ToolRunner.run(new RecordCountDriver(), args);
    System.out.println(jobStatus);
}
```

```
Command Prompt - docker  exec -it hadoop-hive-pig bash
bash-4.1# hdfs dfs -cat output/availabilitycount/part-r-00000
20/01/12 02:45:26 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Count   843
bash-4.1#
```

2.1.2

```
public class RecordCountDriver
        extends Configured implements Tool{
    private  static  class RecordMapper extends Mapper<LongWritable, Text, Text, LongWritable>{

        private Text word = new Text();
        public void map(LongWritable lineOffset, Text record, Context output)
        throws IOException, InterruptedException {
            String row = record.toString();
            String clean_row = row.replaceAll("[\\n\\t ]", "");
            String[] tokens = clean_row.split(",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)");


            String col4 = tokens[4];


            if (!col4.contains("neighbourhood_group")) {
                word.set(col4);
                output.write(word,new LongWritable(1));
            }
            else{
                return;
            }

        }
    }
```

```java
public static class LongSumReducer extends
        Reducer<Text, LongWritable, Text, LongWritable> {
    private LongWritable result = new LongWritable();

    public void reduce(Text key, Iterable<LongWritable> values,
                       Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }

        result.set(sum);
        context.write(key, result);

    }
}
```

```java
public int run(String[] args) throws Exception {
    // TODO Auto-generated method stub
    @SuppressWarnings("deprecation")
    Job job = new Job(getConf(), "Record Count");
    job.setJarByClass(getClass());

    job.setMapperClass(RecordMapper.class);
    job.setReducerClass(LongSumReducer.class);

    job.setNumReduceTasks(1);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(LongWritable.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(LongWritable.class);



    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    return job.waitForCompletion(true) ? 0 : 1;
}

public static void main(String[] args) throws Exception {
    int jobStatus = ToolRunner.run(new RecordCountDriver(), args);
    System.out.println(jobStatus);
}
```

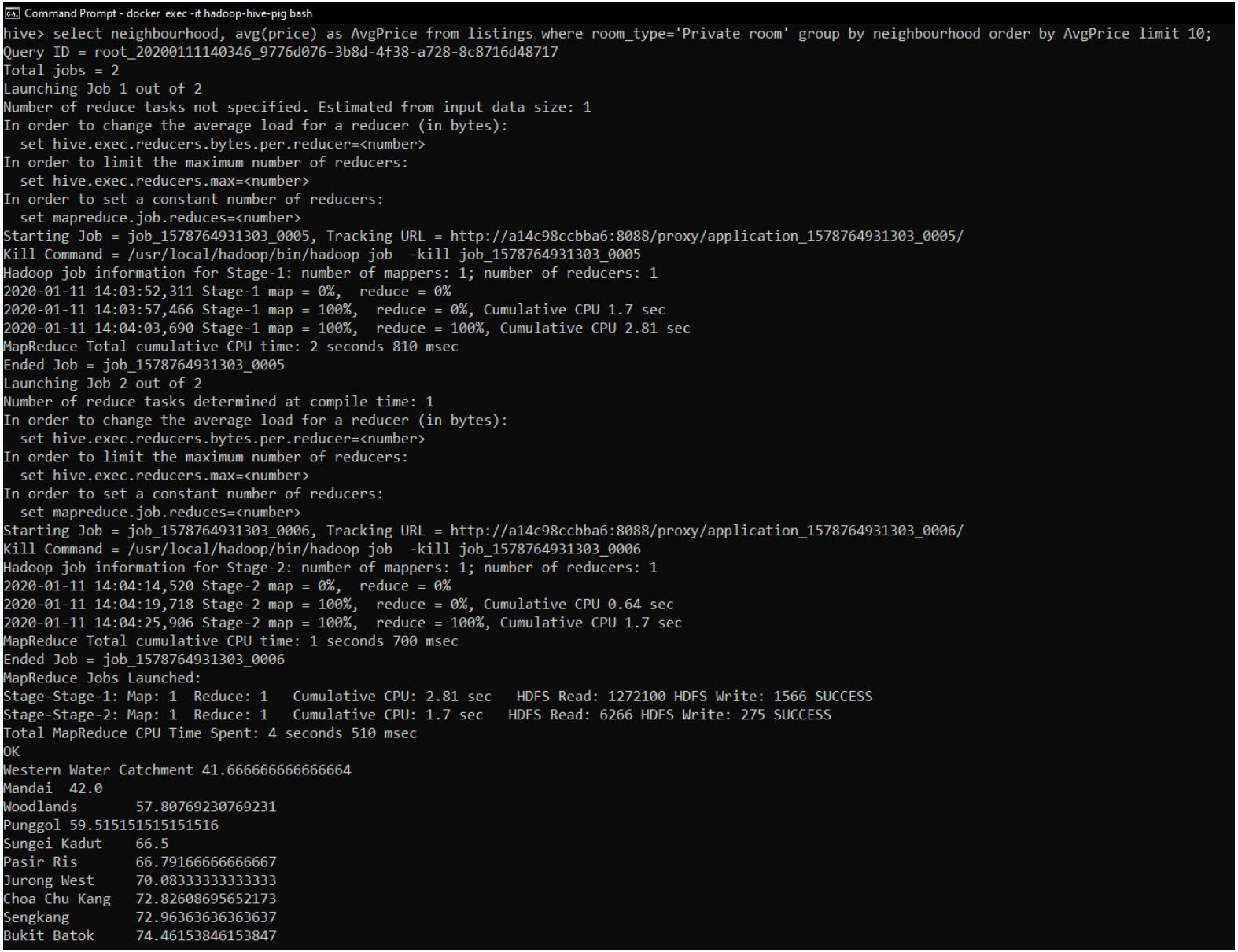
```
Command Prompt - docker  exec -it hadoop-hive-pig bash
bash-4.1# hdfs dfs -cat output/rentalcountbyneighbourhood/part-r-00000
20/01/12 02:55:39 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
"Bedok" 4
"BukitTimah"    2
"CentralRegion" 6295
"EastRegion"    504
"Geylang"       12
"North-EastRegion"      346
"NorthRegion"   204
"WestRegion"    540
bash-4.1#
```

Section 2.2

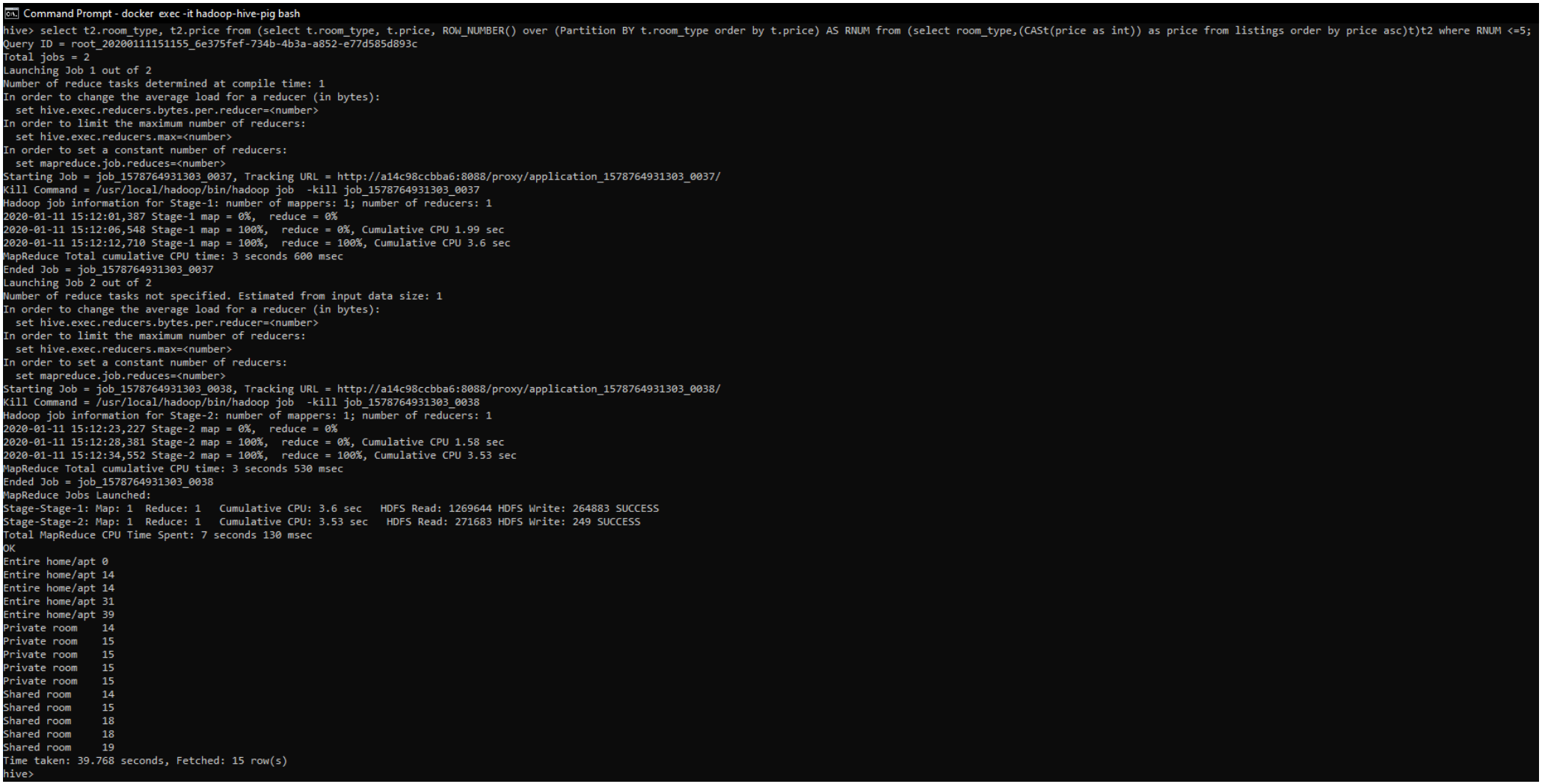
2.2.1

```
hive> select neighbour_group, avg(price) from listings where room_type='Private room' group by neighbour_group;
Query ID = root_20200111135933_88ce2a44-5e7b-44c6-959c-c95037d71149
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1578764931303_0002, Tracking URL = http://a14c98ccbba6:8088/proxy/application_1578764931303_0002/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1578764931303_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2020-01-11 13:59:39,981 Stage-1 map = 0%,  reduce = 0%
2020-01-11 13:59:45,429 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.7 sec
2020-01-11 13:59:52,623 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 2.86 sec
MapReduce Total cumulative CPU time: 2 seconds 860 msec
Ended Job = job_1578764931303_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 2.86 sec   HDFS Read: 1272679 HDFS Write: 163 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 860 msec
OK
Central Region  114.38011695906432
East Region     117.23497267759562
North Region    81.99295774647888
North-East Region       79.87867647058823
West Region     117.82539682539682
Time taken: 20.436 seconds, Fetched: 5 row(s)
hive>
```

2.2.2


Command Prompt - docker exec -it hadoop-hive-pig bash
hive> select neighbourhood, avg(price) as AvgPrice from listings where room_type='Private room' group by neighbourhood order by AvgPrice limit 10;
Query ID = root_20200111140346_9776d076-3b8d-4f38-a728-8c8716d48717
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1578764931303_0005, Tracking URL = http://a14c98ccbba6:8088/proxy/application_1578764931303_0005/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1578764931303_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2020-01-11 14:03:52,311 Stage-1 map = 0%, reduce = 0%
2020-01-11 14:03:57,466 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.7 sec
2020-01-11 14:04:03,690 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.81 sec
MapReduce Total cumulative CPU time: 2 seconds 810 msec
Ended Job = job_1578764931303_0005
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Starting Job = job_1578764931303_0006, Tracking URL = http://a14c98ccbba6:8088/proxy/application_1578764931303_0006/
Kill Command = /usr/local/hadoop/bin/hadoop job -kill job_1578764931303_0006
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2020-01-11 14:04:14,520 Stage-2 map = 0%, reduce = 0%
2020-01-11 14:04:19,718 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.64 sec
2020-01-11 14:04:25,906 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.7 sec
MapReduce Total cumulative CPU time: 1 seconds 700 msec
Ended Job = job_1578764931303_0006
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.81 sec HDFS Read: 1272100 HDFS Write: 1566 SUCCESS
Stage-Stage-2: Map: 1 Reduce: 1 Cumulative CPU: 1.7 sec HDFS Read: 6266 HDFS Write: 275 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 510 msec
OK
Western Water Catchment 41.666666666666664
Mandai 42.0
Woodlands 57.80769230769231
Punggol 59.515151515151516
Sungei Kadut 66.5
Pasir Ris 66.79166666666667
Jurong West 70.08333333333333
Choa Chu Kang 72.82608695652173
Sengkang 72.96363636363637
Bukit Batok 74.46153846153847

2.2.3



```
Command Prompt - docker  exec -it hadoop-hive-pig bash
hive> select t2.room_type, t2.price from (select t.room_type, t.price, ROW_NUMBER() over (Partition BY t.room_type order by t.price) AS RNUM from (select room_type,(CASt(price as int)) as price from listings order by price asc)t)t2 where RNUM <=5;
Query ID = root_20200111151155_6e375fef-734b-4b3a-a852-e77d585d893c
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1578764931303_0037, Tracking URL = http://a14c98ccbba6:8088/proxy/application_1578764931303_0037/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1578764931303_0037
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2020-01-11 15:12:01,387 Stage-1 map = 0%,  reduce = 0%
2020-01-11 15:12:06,548 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.99 sec
2020-01-11 15:12:12,710 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.6 sec
MapReduce Total cumulative CPU time: 3 seconds 600 msec
Ended Job = job_1578764931303_0037
Launching Job 2 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1578764931303_0038, Tracking URL = http://a14c98ccbba6:8088/proxy/application_1578764931303_0038/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1578764931303_0038
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2020-01-11 15:12:23,227 Stage-2 map = 0%,  reduce = 0%
2020-01-11 15:12:28,381 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 1.58 sec
2020-01-11 15:12:34,552 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 3.53 sec
MapReduce Total cumulative CPU time: 3 seconds 530 msec
Ended Job = job_1578764931303_0038
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.6 sec   HDFS Read: 1269644 HDFS Write: 264883 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 3.53 sec   HDFS Read: 271683 HDFS Write: 249 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 130 msec
OK
Entire home/apt 0
Entire home/apt 14
Entire home/apt 14
Entire home/apt 31
Entire home/apt 39
Private room    14
Private room    15
Private room    15
Private room    15
Private room    15
Shared room     14
Shared room     15
Shared room     18
Shared room     18
Shared room     19
Time taken: 39.768 seconds, Fetched: 15 row(s)
hive>
```

Section 2.3

2.3.1

```
%pyspark                                                                    SPARK JOBS FINISHED

from decimal import *

#df = spark.read.csv("/data/listings.csv",header = "true")
pandas_df = pd.read_csv("/data/listings.csv")

pandas_df = pandas_df.replace({r'\\n': ''}, regex=True)

df = sqlContext.createDataFrame(pandas_df.astype(str))

total=df.select(df.host_name,df.calculated_host_listings_count).distinct().orderBy('host_name').count()


d_hn=df.select(df.host_name,df.calculated_host_listings_count).filter(df.calculated_host_listings_count > 1).distinct().orderBy('host_name')
d_hn_c = d_hn.count()
p = round((Decimal(d_hn_c) / Decimal(total))*100,2)
print(d_hn_c)
print(total)
print(p)
```

```
703
2128
33.04
```

Took 7 sec. Last updated by anonymous at January 11 2020, 10:30:25 PM. (outdated)

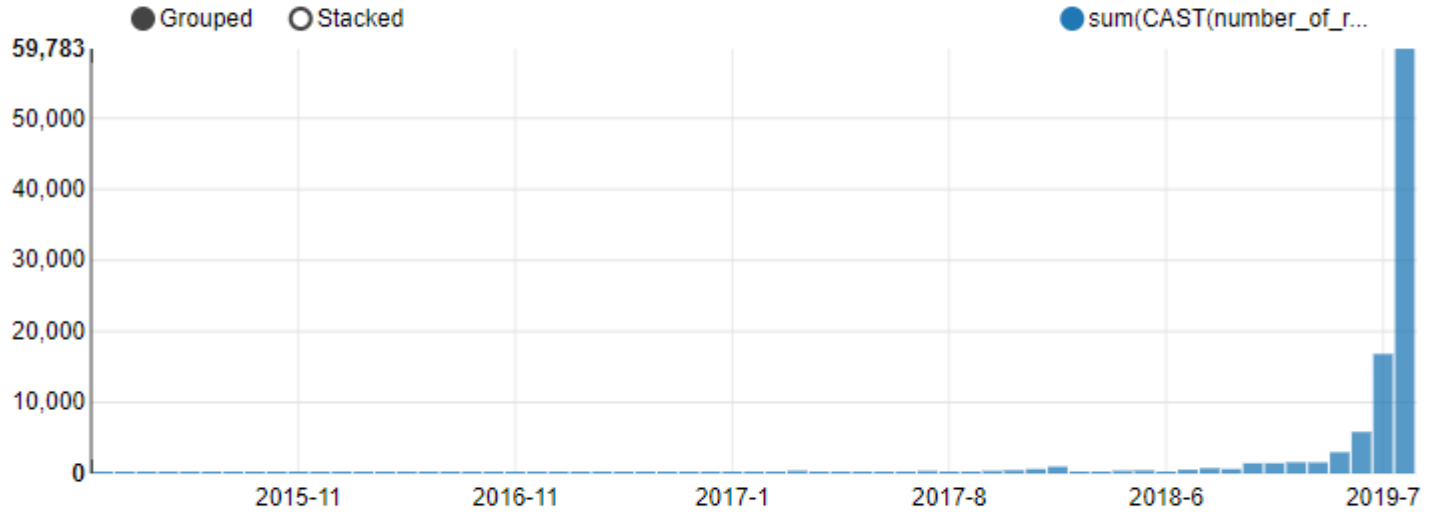
2.3.2

```
%sql

SELECT CONCAT(YEAR(last_review), "-" ,MONTH(last_review)) AS YearMonth, SUM(number_of_reviews )
  FROM listings
  Where last_review >= '2013-10-21' AND last_review != 'nan'
 GROUP BY YearMonth
 Order BY YearMonth
```

SPARK JOB FINISHED

settings



Took 0 sec. Last updated by anonymous at January 11 2020, 10:24:11 PM. (outdated)

2.3.3

```
%pyspark

import shutil
shutil.rmtree('TopFiveNeighbourhoods.parquet', ignore_errors=True)

df = spark.sql("select  neighbourhood,avg(price) AS Price  from listings        Group by neighbourhood    order by price    Limit 5")

df.write.save("TopFiveNeighbourhoods.parquet")

parquetFile = spark.read.load("TopFiveNeighbourhoods.parquet")

df = spark.sql("SELECT count(name) As NumberofListings FROM listings where availability_365 = '365' AND neighbourhood in (select neighbourhood from parquet.`TopFiveNeighbourhoods.parquet`)")

df.show()
```

```
+----------------+
|NumberofListings|
+----------------+
|              22|
+----------------+
```

Section 3

```
%pyspark

import libs

from pyspark.sql import SparkSession, Row, functions, types
from pyspark.sql.functions import udf
import numpy as np
import pandas as pd
from pandas import DataFrame
from pyspark.ml.feature import HashingTF, IDF, Tokenizer, VectorAssembler, StringIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.mllib.regression import LabeledPoint
from pyspark.ml.classification import NaiveBayes
from pyspark.mllib.evaluation import MultilabelMetrics
from pyspark.mllib.linalg import Vectors
from pyspark.ml import Pipeline
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.mllib.evaluation import MulticlassMetrics, BinaryClassificationMetrics
from pyspark.rdd import RDD
from pyspark.sql.functions import col
import pandas as pd
import numpy as np
from pyspark.sql import functions as F
from pyspark.sql.functions import when
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.sql import functions as F
from pyspark.sql.functions import lit
from datetime import date
from pyspark.sql.types import StructField
from pyspark.sql.types import StructType
from pyspark.sql.types import *
from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.mllib.evaluation import MultilabelMetrics
from pyspark.mllib.evaluation import MulticlassMetrics, BinaryClassificationMetrics
from pyspark.rdd import RDD


df = spark.read.csv("/data/listings.csv", header="true", inferSchema="true")
colmns=['id','latitude','longitude','neighbourhood_group']
df_training = df.select(colmns)
df_training = df_training.na.drop()
df_training = df_training.withColumn("longitude", df_training["longitude"].cast(DoubleType()))
df_training.printSchema()

df_training.createOrReplaceTempView("cols")

df1_training = spark.sql("select neighbourhood_group, count(id) id_count from cols group by neighbourhood_group order by id_count DESC")
df1_training.show()

#ignore minimul popular regions
def valueToInt(value):
    if value=='Central Region': return 5
    elif value=='East Region': return 4
```

```
    elif value=='East Region': return 4
    elif value=='North Region': return 3
    elif value=='North-East Region': return 2
    elif value=='West Region': return  1
    else: return 6

udfValueToInt = udf(valueToInt, IntegerType())
df1_training = df_training.withColumn("label_column", udfValueToInt("neighbourhood_group"))

df2_training = df1_training.filter(df1_training['label_column'] < 6)
df2_training.show()

def vectorAssembling(df,impFeatures):
print(" vectorAssembling start.. ")
assembler = VectorAssembler(inputCols=impFeatures,outputCol="features")
df_new=assembler.transform(df)
print("vectorAssembling done.")
return df_new
inputs=['latitude','longitude']
impdf = df2_training.select(impCols).show()
# Create one feature vector column using latitude and longitude
dtf=vectorAssembling(df2_training,inputs)
dtf.show()

finalized_data = dtf.select('label_column', 'features')
from pyspark.ml.feature import StandardScaler
scaler = StandardScaler(inputCol='features', outputCol='scaledFeatures', withStd=True, withMean=True)
scalerModel = scaler.fit(finalized_data)
classiFinalData = scalerModel.transform(finalized_data)

# Split the data into training and test sets (20% held out for testing)
(trainingData, testData) = classiFinalData.randomSplit([0.8, 0.2])

trainingData.show()


def TrainingModel(trainDF):
print("TrainingModel started")
lr = LogisticRegression(labelCol="label_column", featuresCol="scaledFeatures",elasticNetParam=0.8, family="multinomial",maxIter=100)
model=lr.fit(trainDF)
print("TrainingModel completed")
return model

model = TrainingModel(trainingData)

# Get predictions.
predictions = model.transform(testData)

# Predicted data
predictions.select("prediction", "label_column", "features").show(100)




from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.mllib.evaluation import MultilabelMetrics
from pyspark.mllib.evaluation import MulticlassMetrics, BinaryClassificationMetrics
from pyspark.rdd import RDD

evaluatorRecall = MulticlassClassificationEvaluator(labelCol="label_column", predictionCol="prediction", metricName="weightedRecall")
evaluatorPrecision = MulticlassClassificationEvaluator(labelCol="label_column", predictionCol="prediction", metricName="weightedPrecision")
recall = evaluatorRecall.evaluate(predictions)
precision = evaluatorPrecision.evaluate(predictions)
print("Recall %s" % recall)
print("Precision %s" % precision)
```

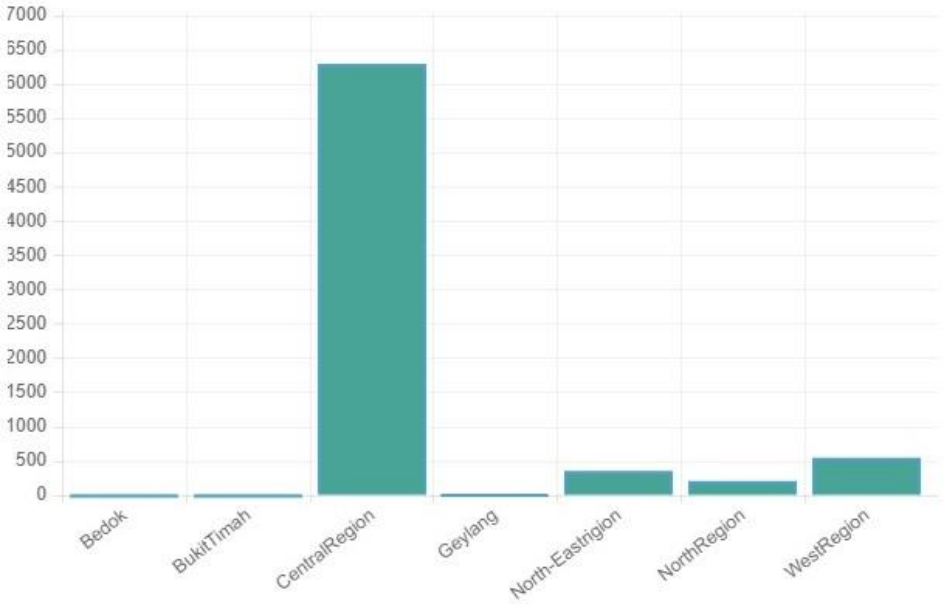
Section 4

4.1 & 4.2

1. Total number of rentals available 365 days a year

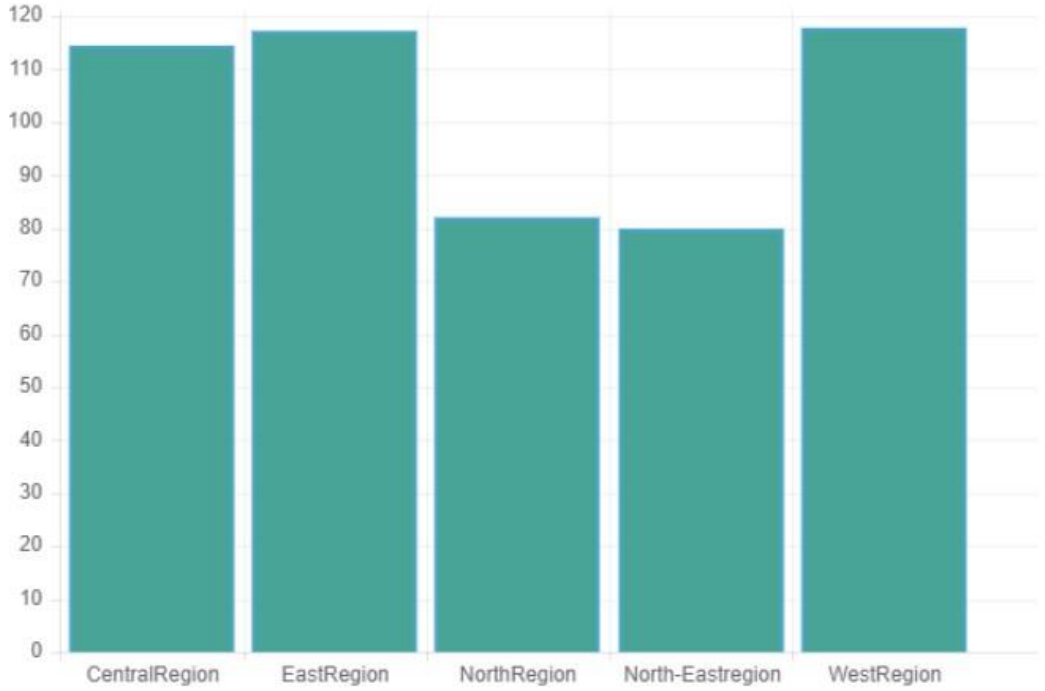
843

2. Numbers of Rentals per neighbourhood_group



4.3

3. Avareage Price of private room Rentals by neighbourhood_group



4.4

4. Neighbourhood based on Average price of Private room

neighbourhood	Average price of Private room
Western Water Catchment	41.66
Mandai	42.0
Woodlands	57.80
Punggol	59.51
Sungei Kadut	66.50
Pasir Ris	66.79
Jurong West	70.08
Choa Chu Kang	72.82
Sengkang	72.96
Bukit Batok	74.46

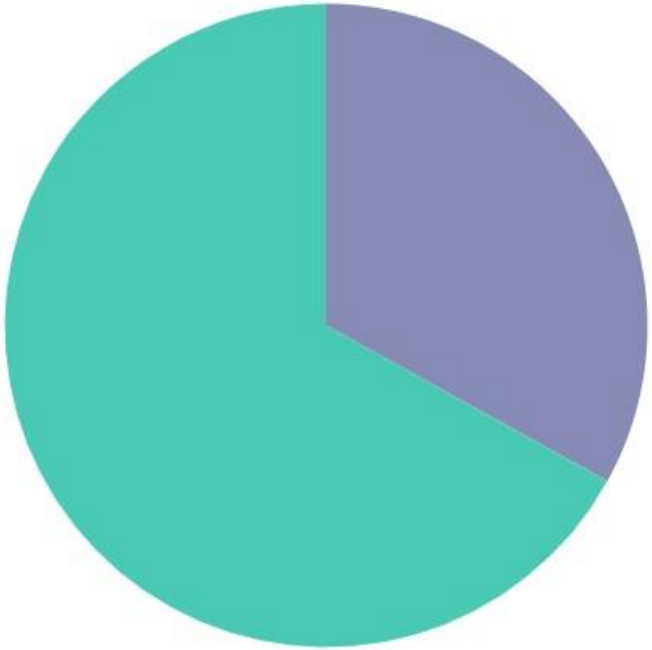
4.5

5. lowest price rentals per each room_type

Room Type	Price
Entire home/apt	0
Entire home/apt	14
Entire home/apt	14
Entire home/apt	31
Entire home/apt	39
Private Room	14
Private Room	15
Private Room	15
Private Room	15
Private Room	15
Private Room	14
Private Room	15
Private Room	18
Private Room	18
Private Room	19

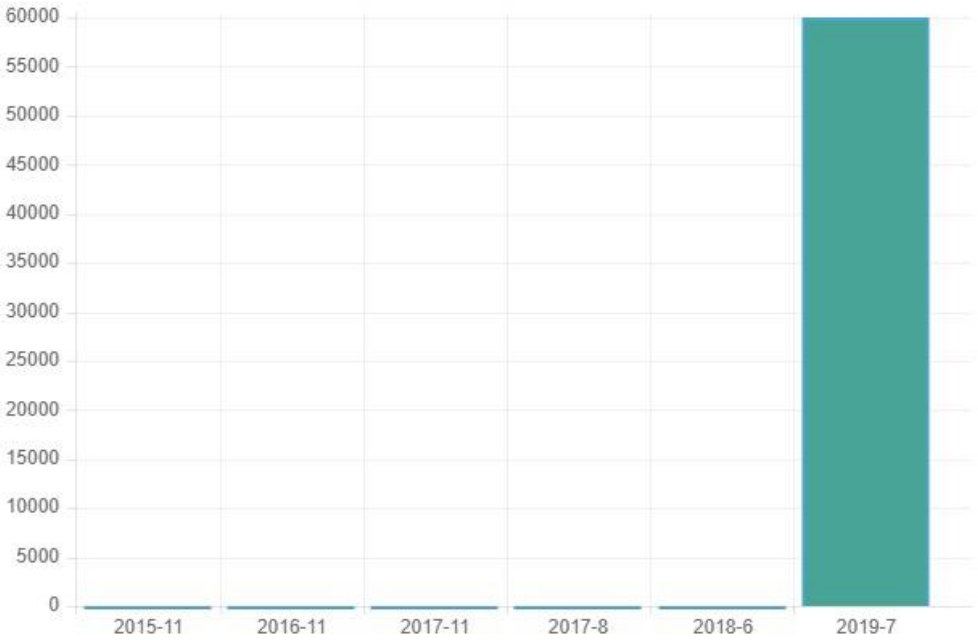
4.6

6. Owners who rent more than one property



4.7

7. Histogram of number of rentals reviewed over time (based on last_review) in mouth granularity



Appendix

Source Code Repository

https://github.com/gotukola/hive-spark-mapreduce-javascript