

**MSc in**

**Big Data Analytics**

|  |  |  |
| --- | --- | --- |
| Student Name:  Anuradha Nanayakkara | RGU Number:  1815303 | |
| IIT Number:  2019009 | Module:  Big Data Programming | |
| Course Work No:  CMM705 Big Data Programming | | |
| Course Work Title | Due Date:  2020/01/12 | |
| Submission Date:  2020/01/09 | |
| **CONSENT**  x  I agree  I do not agree  That the University shall be entitled to use any results, materials or other outcomes arising from my project work for the purposes of non-commercial teaching and research, including collaboration.  **DECLARATION**  I confirm:   1. That the work contained in this document has been composed solely by myself and that I have not made use of any unauthorised assistance. 2. That the work has not been accepted in any previous application for a degree. 3. All sources of information have been specifically acknowledged and all verbatim extracts are distinguished by quotation marks. | | |
| Student Signature:  Anuradha Nanayakkara | | Date Signed:  2020/01/09 |

Table of Contents

[Part 1 3](#_Toc29416704)

[Deployment Architecture 3](#_Toc29416705)

[Part 2 5](#_Toc29416706)

[Map Reduce Jobs 5](#_Toc29416707)

[2.1.1 Total number of rentals that are available 365 days a year 5](#_Toc29416708)

[2.1.2. Number of rentals per neighborhood group 9](#_Toc29416709)

[13](#_Toc29416710)

[Hive Queries: 14](#_Toc29416711)

[2.2.1 Average price of Private room rental by neighbourhood\_group. 14](#_Toc29416712)

[2.2.2 Top 10 neighborhood based on Average price of Private room. 15](#_Toc29416713)

[2.2.3 The 5 lowest price properties per each room\_type 16](#_Toc29416714)

[Spark Queries 18](#_Toc29416715)

[2.3.1 Percentage of owners who rent more than one property. 18](#_Toc29416716)

[2.3.2 Histogram of number of rentals reviewed over time (based on last review) in mouth granularity 18](#_Toc29416717)

[2.3.3 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. 22](#_Toc29416718)

[Part 3 23](#_Toc29416719)

[3.1 MLlib and build a classification model that predict the neighbourhood\_group 23](#_Toc29416720)

[Steps: 25](#_Toc29416721)

[Part 4 26](#_Toc29416722)

[Screenshot of the dashboard 26](#_Toc29416723)

# Part 1

## Deployment Architecture

According the above diagram there are few layers which are support for the collect and analyze data thought a Batch processing system in the Singapore Airbnb.

**Data Sources: -** Data flowing to the system from various data sources like CSV and OLTP. So, the data can be varying . In our solutions data source was a CSV file. I have assumed that the data was a in structured format.

**Data ingestion Layer**: - This layer accepts the data from different variable sources and data prioritization and categorization is done by this layer which support the easy access data for the other layers.

1. For the data in ingestion you can use the apache **Sqoop** or **Nifi** to transfer data between relational database and the Hadoop File System. If you have unstructured data, you can use **Flume** as well. Since the Airbnb have the structured data, we can use above mention Sqoop or **Nifi**. **Sqoop** has command line interface and the bidirectional data flow while **Nifi** has the GUI with more functionality like security scalability integrated with it.

**Data Storage Layer: -** Before the Processing and after the processing of data need to store in the scalable and reliable storage this layer support for store data in more efficient way.

1. This layer is responsible for the storing data in the distributed file system like **HDFS** before processing and stored the data in **NoSQL** database after processing. **HDFS** store all the data flowing through the Ingestion layer and after all the data are being processed and then data will store in the **Cassandra** or **HBase** NoSQL Databases.

**Data Procession Layer** & **Query Layer**: - This Layer is hybrid version of the Batch Processing and Query Layer in the Big Data Architecture. According to the project its needs only the batch processing not real time processing, and this is not processing the data in real time (No need of the Streaming).

1. **Batch processing layer** focus only collected the data from previous layers pipeline the processing system. This is the place where the analytics of data begin.
2. **Query Layer** focus on the analytics. This is the layer that analyzing part is actively work on.

For the batch procession it takes the chuck of data at once and process it from batch wise and write it at once to the **HDFS** as large output. best batch processing technique is **Hadoop** **Map Reduce**. In the query layer we can use the **Hive**, **Pig** or the **spark** for the analyze the large set of datasets. **Hive** is doing the summarization, and this is very similar to SQL language call **HiveQL**. **Apache Spark** also a query language which supportive for the cost base optimizer. After getting output of the quires data is store in the **HBase** or the **Casandra** **NoSQL** database. For this project I have preferred **Hive**, **Spark** SQL and **HBase** database

**Analytics and Machine Learning**: - This part solely responsible for the Machine learning which take the data form the Query Layer and take the analytical prediction form it. In the program.

1. We can use **Spark MLlib** or **Mahout** for the machine Learning. Using the processed data, we can analyze the prediction based on the results. For this I have used the **Spark MLlib** for the Machine learning.

**Search and Indexing:** - This part is supporting for the searching text and indexing the queries related to the analytics.

1. For this I have used the **Apache solar** which is supportive for the full-text search, document handling and the indexing.

**Visualization Layer**: - This layer focus on the visualizing the analytical data on the static web page dashboards or the software’s like **tableau**, **power BI** or **Katana**

1. This can be real time visualization dashboards or any other custom dashboards. After all the flows finally get in to the output section in the environment for the visualization we can get the data from the **Cassandra** or **HBase** from NoSQL and we can generate the reports based on the data. As I mention above you can use **Power BI, Katana and or Tubule**.

**Data Security Layer:** - Every layer need security from starting from ingestion layer. There are different ways to secure the system Authentication, **Access Control, Encryption and Data Masking** are the few of them.

**Data Monitoring Layer**: - This is done all the Data Profiling and Linage, Data Quality checking, Data Cleansing, Data Preventions. This is ensuring the data quality of each layer and remove all the corrupted data and do the quality assessment.

# Part 2

## Map Reduce Jobs

## 2.1.1 Total number of rentals that are available 365 days a year

#### Configuration

1. docker run -it -p 8100:8088 -p 53800:50070 --name Assignment1 -v D:\BigDataProgram\BigDataAnalytics\Assignment\Quection2.1\ProjectResource:/resource -d suhothayan/hadoop-spark-pig-hive:2.9.2
2. docker exec -it Assignment1 bash
3. sudo apt-get install unzip
4. cd resource
5. hdfs dfs -put stackapps.com stackapps.com
6. yarn jar Airbnb/target/Airbnb\_Rental.jar com.airbnb.mr.RentalCount stackapps.com/comments /output/RentalCount

package com.airbnb.mr;

import java.io.IOException;

import java.util.\*;

import org.apache.commons.lang.StringEscapeUtils;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;

public class RentalCount {

public static class WordCountMapper extends Mapper<Object, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

final String DELIMITER = ",";

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String line = value.toString();

if(line.toString().startsWith("id,name")){

// Skip header line (first line) of CSV

return;

}

StringBuilder sb = new StringBuilder();

String[] tokens = value.toString().split(DELIMITER);

int totalError=0;

for(int i=0; i< tokens.length;i++){

String token = tokens[i];

int totalLength = token.length();

token=token.trim();

token = token.replaceAll("^\"|\"$", "");

totalError+=(totalLength-token.length());

sb.append(token);

if(i<tokens.length-1)

sb.append(DELIMITER);

}

line=sb.toString();

int calculatedhostListing=0;

String data[] = line.toString().split(",", -1);

if(data.length==16) {

if(data[14].trim()!="")

calculatedhostListing= Integer.parseInt(data[14]);

else

calculatedhostListing=1;

String rentals = data[15];

if( Integer.parseInt(data[15]) ==365) {

word.set(rentals);

one.set(1);

context.write(word, one);

}

}

}

}

public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

result.set(sum);

context.write(key, result);

}

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

String[] otherArgs = new GenericOptionsParser(conf, args)

.getRemainingArgs();

Job job = Job.getInstance(conf, "word count");

job.setJarByClass(RentalCount.class);

job.setMapperClass(WordCountMapper.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

FileOutputFormat.setOutputPath(job, new Path(args[2]));

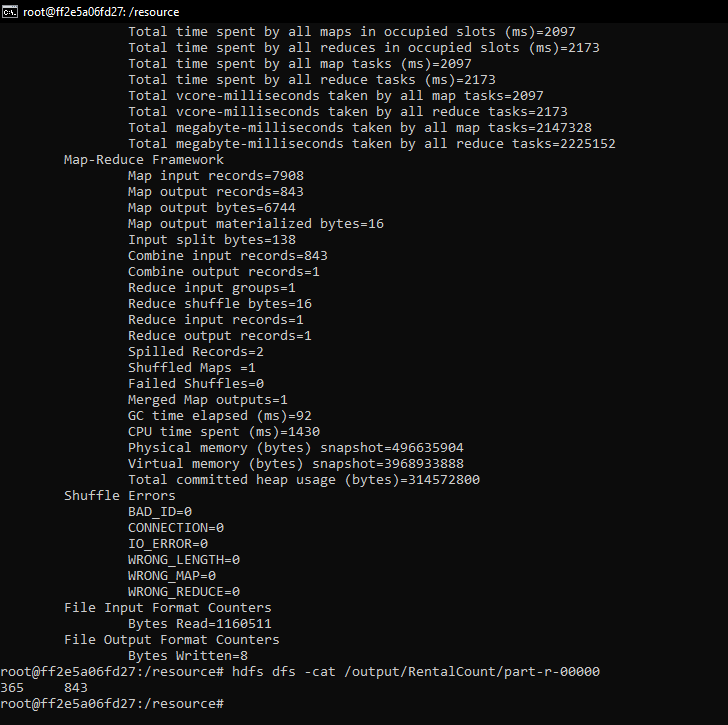
System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

#### Result:

hdfs dfs -cat /output/RentalCount/part-r-00000



## 2.1.2. Number of rentals per neighborhood group

#### Configuration

As same Configuration in First Map Reduce Job

1. yarn jar Airbnb/target/Airbnb\_Ne.jar com.airbnb.mr.NeighbourhoodGroupCount stackapps.com/comments /output/Neghood

package com.airbnb.mr;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;

import java.io.IOException;

public class NeighbourhoodGroupCount {

public static class WordCountMapperNe extends Mapper<Object, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

final String DELIMITER = ",";

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String line = value.toString();

if(line.toString().startsWith("id,name")){

// Skip header line (first line) of CSV

return;

}

StringBuilder sb = new StringBuilder();

String[] tokens = value.toString().split(DELIMITER);

int totalError=0;

for(int i=0; i< tokens.length;i++){

String token = tokens[i];

int totalLength = token.length();

token=token.trim();

token = token.replaceAll("^\"|\"$", "");

token =token.replace("\n", " ");

totalError+=(totalLength-token.length());

sb.append(token);

if(i<tokens.length-1)

sb.append(DELIMITER);

}

line=sb.toString();

String data[] = line.toString().split(",", -1);

if(data.length==16) {

String neighbourhood\_group = data[4];

word.set(neighbourhood\_group);

one.set(1);

}

context.write(word, one);

}

}

public static class IntSumReducerNe extends Reducer<Text, IntWritable, Text, IntWritable> {

private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

result.set(sum);

context.write(key, result);

}

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

String[] otherArgs = new GenericOptionsParser(conf, args)

.getRemainingArgs();

Job job = Job.getInstance(conf, "negi");

job.setJarByClass(NeighbourhoodGroupCount.class);

job.setMapperClass(WordCountMapperNe.class);

job.setCombinerClass(IntSumReducerNe.class);

job.setReducerClass(IntSumReducerNe.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

FileOutputFormat.setOutputPath(job, new Path(args[2]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

#### Result:

hdfs dfs -cat /output/Neghood/part-r-00000

### 

## Hive Queries:

#### Configuration:

1. docker run -it -p 8100:8088 -p 53800:50070 --name Assignment1 -v D:\BigDataProgram\BigDataAnalytics\Assignment\Quection2.1\ProjectResource:/resource -d suhothayan/hadoop-spark-pig-hive:2.9.2
2. docker exec -it Assignment1 bash
3. hdfs dfs -put stackapps.com stackapps.com
4. hdfs dfs -mkdir /hiveinput
5. hdfs dfs -put stackapps.com/comments/listingsnew.csv /hiveinput/
6. hdfs dfs -ls /
7. hive

#### Creating Schema and table.

CREATE SCHEMA IF NOT EXISTS bnb;

CREATE EXTERNAL TABLE IF NOT EXISTS bnb.airbnb\_csv\_table

(id STRING,name STRING,host\_id STRING,host\_name STRING,neighbourhood\_group STRING,neighbourhood STRING,latitude STRING,longitude STRING,room\_type STRING,price FLOAT,minimum\_nights STRING,number\_of\_reviews STRING,last\_review STRING,reviews\_per\_month STRING,calculated\_host\_listings\_count STRING,availability\_365 STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

LOCATION '/hiveinput/';

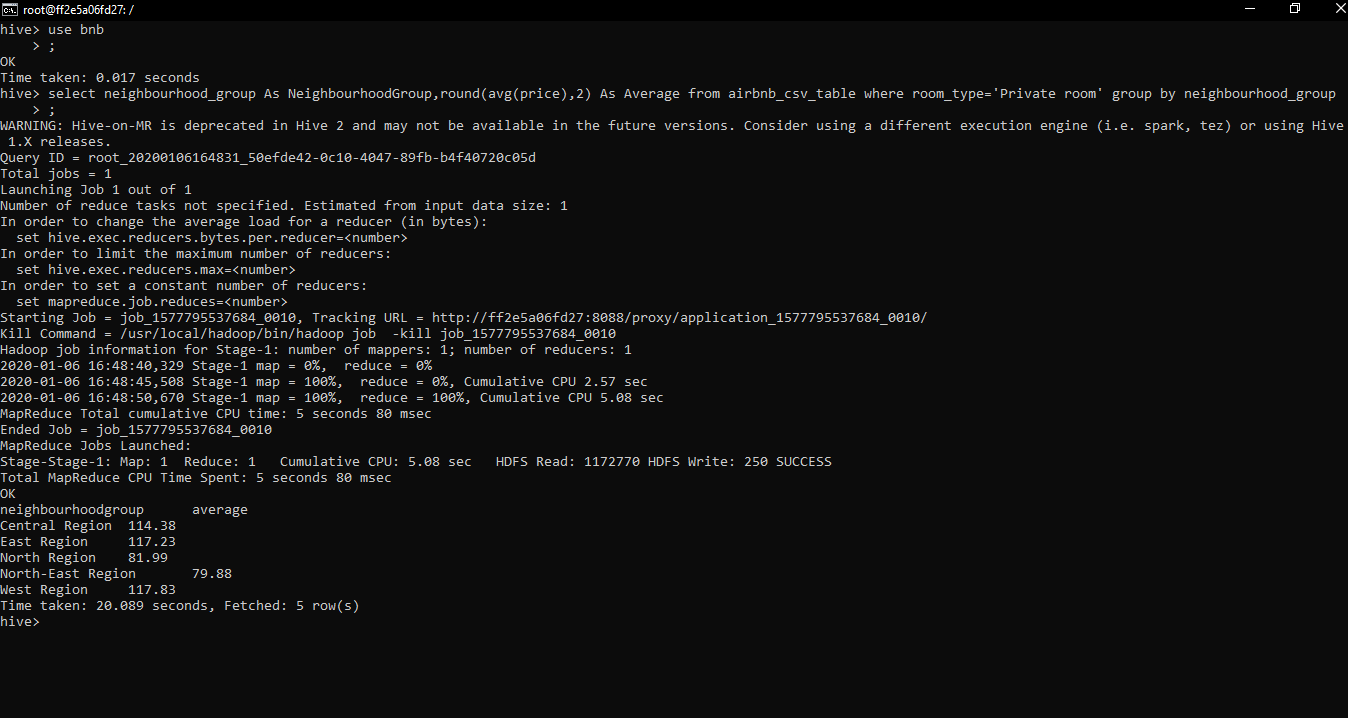
set hive.cli.print.header=true;

### 2.2.1 Average price of Private room rental by neighbourhood\_group.

select neighbourhood\_group As NeighbourhoodGroup,round(avg(price),2) As Average from airbnb\_csv\_table where room\_type='Private room' group by neighbourhood\_group

INSERT OVERWRITE DIRECTORY '/hiveout/average' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' (select neighbourhood\_group As NeighbourhoodGroup,round(avg(price),2) As Average from airbnb\_csv\_table where room\_type='Private room' group by neighbourhood\_group);

#### Result:

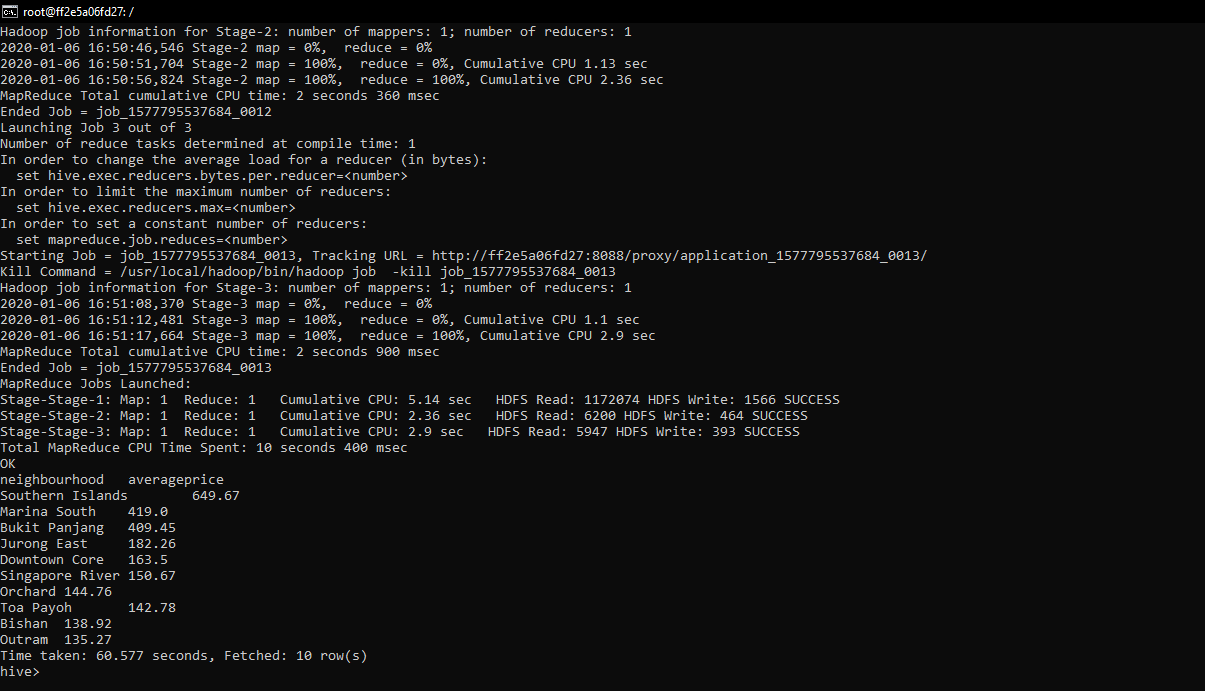
hdfs dfs -copyToLocal /hiveout/average/000000\_0 /resource/1\_Hive\_averagePriceroom\_ngb.csv

### 2.2.2 Top 10 neighborhood based on Average price of Private room.

select neighbourhood As Neighbourhood,round(avg(price),2) as AveragePrice from airbnb\_csv\_table where room\_type='Private room' group by neighbourhood sort by averageprice desc LIMIT 0,10

INSERT OVERWRITE DIRECTORY '/hiveout/averagetopten' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' (select neighbourhood As Neighbourhood,round(avg(price),2) as AveragePrice from airbnb\_csv\_table where room\_type='Private room' group by neighbourhood sort by averageprice desc LIMIT 0,10);

#### Result:

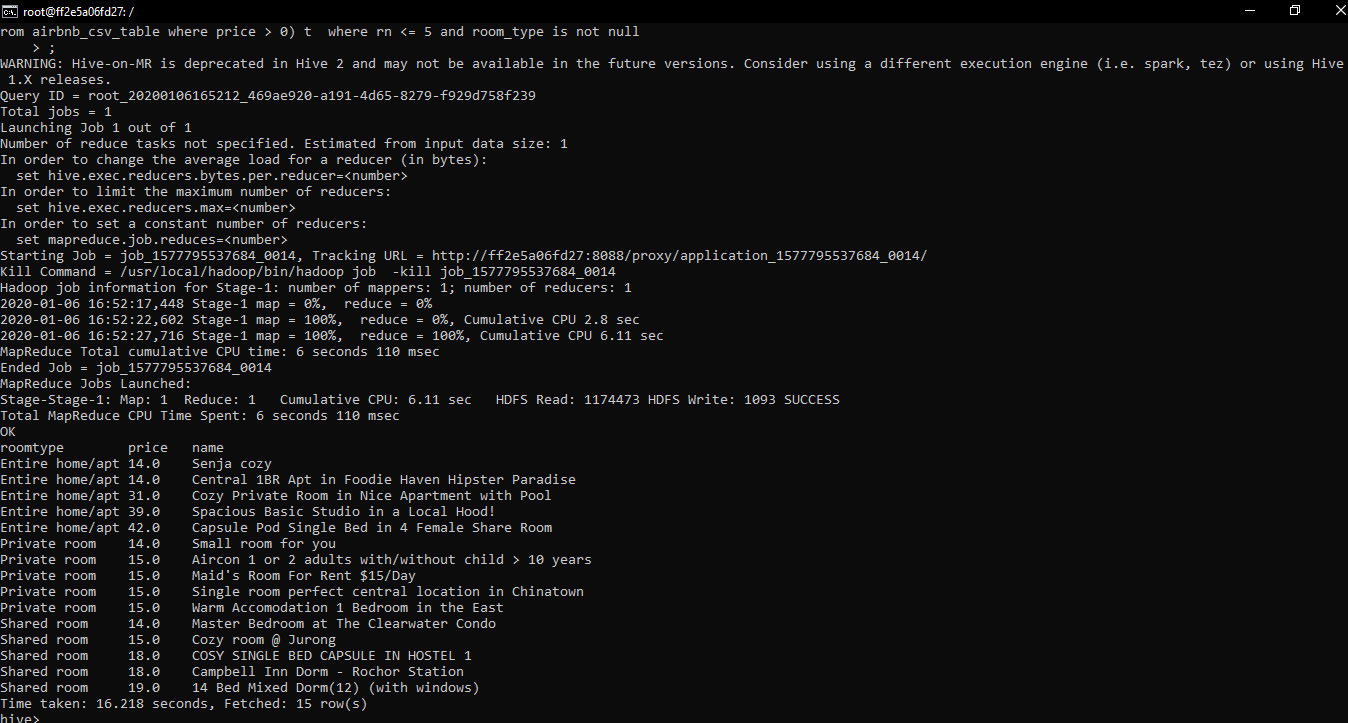
hdfs dfs -copyToLocal /hiveout/averagetopten/000000\_0 /resource/2\_Hive\_Top\_ten\_negb.csv

### 2.2.3 The 5 lowest price properties per each room\_type

select room\_type as RoomType,price as Price,name As Name from (select room\_type, price , name, ROW\_NUMBER() over ( partition by room\_type order by price) as rn from airbnb\_csv\_table where price > 0) t where rn <= 5 and room\_type is not null

INSERT OVERWRITE DIRECTORY '/hiveout/minimumprice' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' (select room\_type as RoomType,price as Price,name As Name from (select room\_type, price , name, ROW\_NUMBER() over ( partition by room\_type order by price) as rn from airbnb\_csv\_table where price > 0) t where rn <= 5 and room\_type is not null);

#### Result:

hdfs dfs -copyToLocal /hiveout/minimumprice/000000\_0 /resource/3\_Hive\_5\_lowersprice.csv

## Spark Queries

### 2.3.1 Percentage of owners who rent more than one property.

%pyspark

import pyspark.sql.functions as func

from pyspark.sql.types import \*

df = spark.read.csv("/data/listingsnew.csv", header=True)

df = df.withColumn("price", df["price"].cast(DoubleType()))

df.createOrReplaceTempView("airbnb")

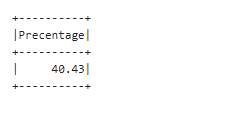
sqlDF = spark.sql("SELECT (Count(\*) \* 100/ (Select Count(distinct host\_name) From airbnb)) As Precentage From (SELECT Count(distinct host\_name),host\_name FROM airbnb Group by host\_name HAVING Count(host\_name) >1)")

sqlDF=sqlDF.withColumn("Precentage", func.round(sqlDF["Precentage"],2))

sqlDF.coalesce(1).write.mode("append").json("/data/output/1\_precentage\_more\_than\_1\_property")

sqlDF.show(15000)

#### Result :



### 2.3.2 Histogram of number of rentals reviewed over time (based on last review) in mouth granularity

%pyspark

import matplotlib.pyplot as plt

import matplotlib.pyplot as plt2

from pyspark.sql.functions import month

import pandas as pd

import numpy as np

import datetime

from matplotlib.ticker import FuncFormatter

import calendar

import datetime

%matplotlib inline

#############################Bar Plot For Monthly Reviews ############################################################

df = spark.read.csv("/data/listingsnew.csv", header=True)

df =df.withColumn("last\_review",month(df["last\_review"]))

df.createOrReplaceTempView("airbnb")

sqlDF = spark.sql("SELECT \* FROM (SELECT last\_review as Last\_Review\_Month,sum(number\_of\_reviews) as NumberOfReviews FROM airbnb where last\_review is not null group by last\_review order by last\_review)")

sqlDF.coalesce(1).write.mode("append").json("/data/output/2\_rentals\_reviewed\_over\_time\_json")

xmonths=[]

yReviews=[]

monthlist = sqlDF.rdd.map(lambda p: p.Last\_Review\_Month).collect()

for month in monthlist:

xmonths.append(calendar.month\_abbr[month])

reviewlist = sqlDF.rdd.map(lambda p: p.NumberOfReviews).collect()

for review in reviewlist:

yReviews.append(review)

fig, ax = plt.subplots()

index = np.arange(12)

bar\_width = 0.8

opacity = 0.5

rects1 = plt.bar(index, yReviews, bar\_width,

alpha=opacity,

color='b',

label='Reviews')

plt.xlabel('Months') #xlabel

plt.ylabel('Reviews') #y alable

plt.title('Monthly Analysis for Reviews') #Heading

plt.xticks(index, xmonths)

plt.legend()

plt.tight\_layout()

plt.show()

###############Histrogram Creation############################################

df2 = spark.read.csv("/data/listingsnew.csv", header=True)

df2.createOrReplaceTempView("airbnbHistro")

sqlDF1 = spark.sql("SELECT last\_review FROM airbnbHistro where last\_review is not null")

monthlistval = sqlDF1.rdd.map(lambda p: p.last\_review).collect()

dtm = lambda x: int(x[5:7])

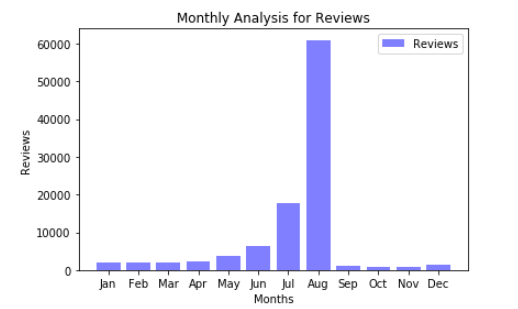
months = list(map(dtm, monthlistval))

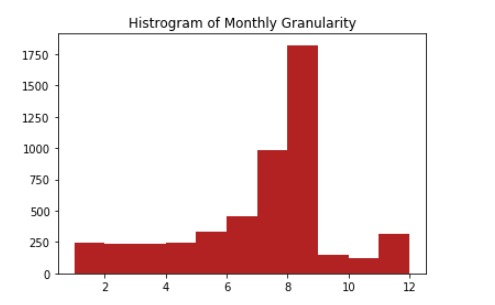
plt2.title('Histrogram of Monthly Granularity')

plt2.hist(months, bins=[1,2,3,4,5,6,7,8,9,10,11,12],color = "firebrick")

##########################################################################

#### Result:





### 2.3.3 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.

#### Query:

%pyspark

from pyspark.sql.types import \*

df = spark.read.csv("/data/listingsnew.csv", header=True)

df = df.withColumn("availability\_365", df["availability\_365"].cast(IntegerType()))

df = df.withColumn("price", df["price"].cast(DoubleType()))

df.createOrReplaceTempView("airbnb")

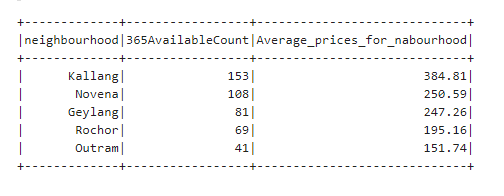
sqlDF = spark.sql("SELECT neighbourhood as neighbourhood,Count(availability\_365) as 365AvailableCount ,((sum(price)\*100)/(Select count(\*) from airbnb)) as Average\_prices\_for\_nabourhood FROM airbnb where availability\_365=365 and price is not NULL group by neighbourhood order by Average\_prices\_for\_nabourhood desc Limit 5")

sqlDF=sqlDF.withColumn("Average\_prices\_for\_nabourhood", func.round(sqlDF["Average\_prices\_for\_nabourhood"],2))

sqlDF.show(8000)

sqlDF.write.format("com.databricks.spark.csv").save("/data/output/3\_Spartk\_rentals\_that\_are\_available\_all\_365.csv")

#### Result:



# Part 3

### 3.1 MLlib and build a classification model that predict the neighbourhood\_group

%pyspark

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.regression import LinearRegression

from pyspark.ml.regression import RandomForestRegressor

from pyspark.ml import Pipeline

from pyspark.ml.evaluation import RegressionEvaluator

from pyspark.sql import functions as f

from pyspark.ml.feature import VectorIndexer

company\_df = sqlContext.read.format("com.databricks.spark.csv").options(header="true", inferschema="true").load("/data/listingsnew.csv")

company\_df=company\_df.withColumn("neighbourhood\_group\_number", f.when(f.col("neighbourhood\_group") == "North Region",0)

.when(f.col("neighbourhood\_group") == "Central Region",1)

.when(f.col("neighbourhood\_group") == "East Region",2)

.when(f.col("neighbourhood\_group") == "North-East Region",3)

.when(f.col("neighbourhood\_group") == "West Region",4)

.otherwise(5))

vectorAssembler = VectorAssembler(inputCols = ["neighbourhood\_group\_number", "latitude","longitude"], outputCol = "features")

tcompany\_df = vectorAssembler.transform(company\_df)

tcompany\_df = tcompany\_df.select(["features", "neighbourhood\_group\_number"])

#tcompany\_df.show(3)

featureIndexer =VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=5).fit(tcompany\_df)

rf = RandomForestRegressor(featuresCol="indexedFeatures",labelCol="neighbourhood\_group\_number")

# Split data into training (80%) and test (20%)

training, test = tcompany\_df.randomSplit([0.8, 0.2], seed=11)

training.cache()

pipeline = Pipeline(stages=[featureIndexer, rf])

model = pipeline.fit(training)

# Make predictions.

predictions = model.transform(test)

# Select example rows to display.

newDataFream=predictions.select("prediction", "neighbourhood\_group\_number", "features")

newDataFream=newDataFream.withColumn("neighbourhood\_group\_number", f.when(f.col("neighbourhood\_group\_number") == 0,"North Region")

.when(f.col("neighbourhood\_group\_number") == 1,"Central Region")

.when(f.col("neighbourhood\_group\_number") == 2,"East Region")

.when(f.col("neighbourhood\_group\_number") == 3,"North-East Region")

.when(f.col("neighbourhood\_group\_number") == 4,"West Region"))

newDataFream.show(10)

# Select (prediction, true label) and compute test error

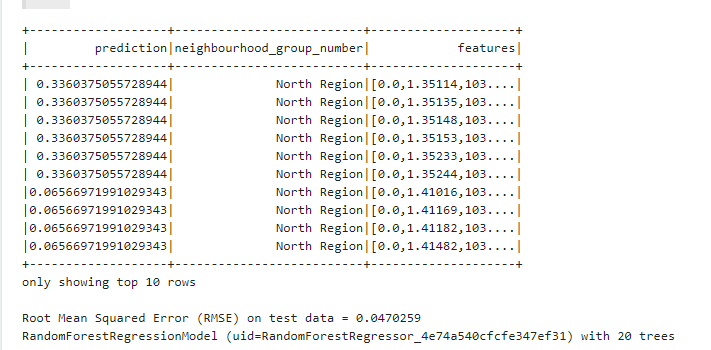
evaluator = RegressionEvaluator(labelCol="neighbourhood\_group\_number", predictionCol="prediction", metricName="rmse")

rmse = evaluator.evaluate(predictions)

print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)

rfModel = model.stages[1]

print(rfModel)



### Steps:

I have used the RandomForestRegressor Classification Model for the Since it has more than two Classified Groups.

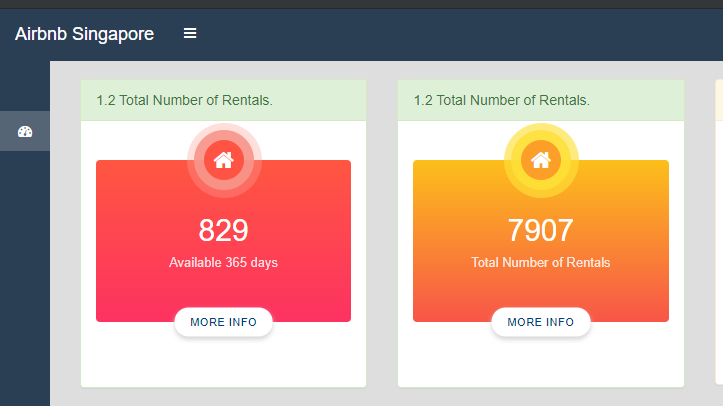
1. Retrieved the neighbourhood\_group from the dataset and assign the values to the each group since it has accepts value for the vectorAssembler.
2. Assign the Vector Assembler input columns as VectorAssembler(inputCols = ["neighbourhood\_group\_number", "latitude","longitude"], outputCol = "features") return output column as features
3. Transform the all the values to binaries.
4. Create feature Indexes in binaries set maximum category number is 5 and fit

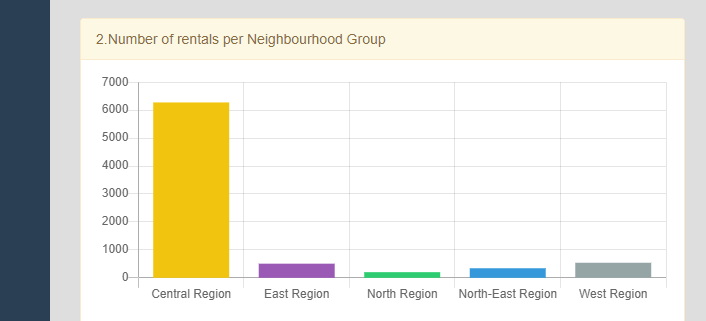
featureIndexer =VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=5).fit(tcompany\_df)

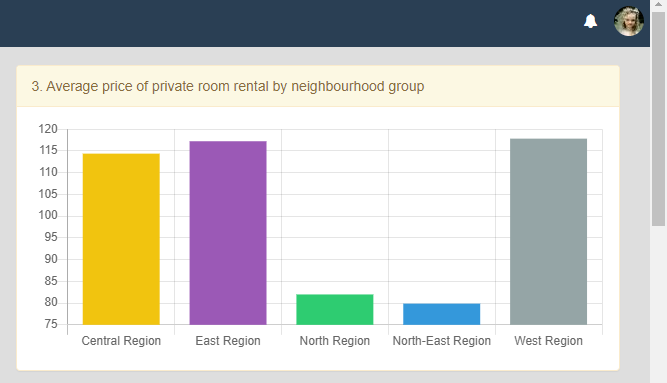
1. Set the RandomForest regressior and Split data into training (80%) and test (20%) With 11 seeds
2. Fit the training features and make prediction using test data.
3. Again, converted to the neighbourhood\_group to orginal names;
4. Using RegressionEvaluator Calculate the mean squre Error.
5. Print the RandomForestRegressionModel its showed the 20 trees.

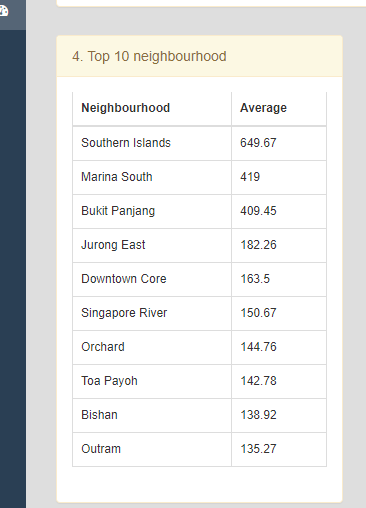
# Part 4

### Screenshot of the dashboard

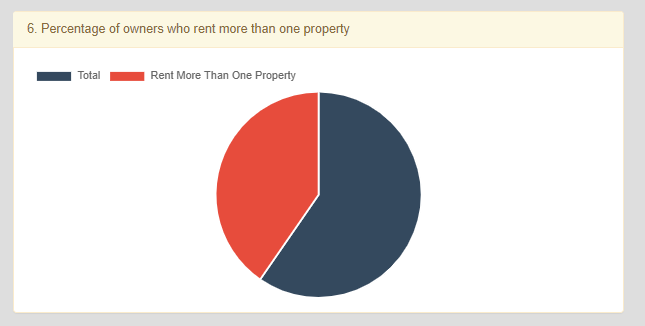


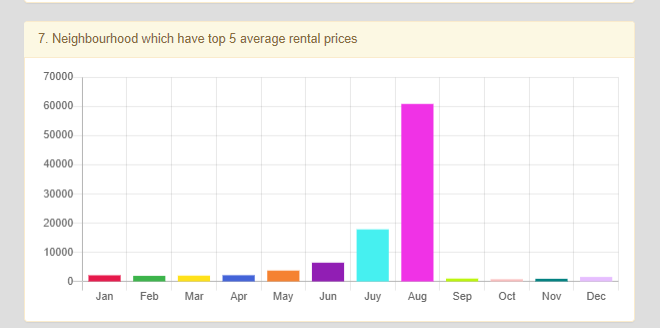














All Pages of Dashboard

