INFO 7250 Engineering of Big-Data Systems

Analysis of Airbnb Rentals

- Urvashi Jain

Introduction

Performed analysis of Airbnb Rental listings of the Boston region to study its growth patterns and impact of various factors on its bookings using MongoDB, Hadoop Mapreduce, Hive and Pig. Mahout Recommendation engine is used to provide recommendations for users.
Few key analysis performed include:

- How popular Airbnb has become over the years?
- Are the demand and prices of rentals correlated?
- What are different types of rentals available in Boston?
- What are highly rated neighborhoods/localities in Boston?
- What is the dependency of host attributes on bookings?
- How cancellation policies vary according to property types?

Dataset details:

The dataset is available at http://insideairbnb.com/get-the-data.html and comprises of three main files:

- listings - Detailed listings data showing 83 attributes for each of the listings. Some of the attributes used in the analysis are listing_id, host_id, host_response_time, host_response_rate, listing_type, neighbourhood, rating_scores_location, rating_scores_review among others.
- reviews - Detailed reviews given by the guests with 6 attributes. Key attributes include date, listing_id, and reviewer_id.
- calendar - Detailed booking data for the 2019 and 2020 for each listing. Key attributes include listing_id, date, available and price.

Steps to start MongoDB:

- Navigate to MONGO_HOME directory and run mongod.exe to start the daemon.
- Once the daemon is started, open another shell and again navigate to MONGO_HOME directory and run mongo.exe to enter the mongo shell.

Steps to start Hadoop Distributed File System:

- Open Terminal and run ssh localhost to set up connection with localhost.
- Then Navigate to HADOOP_HOME directory and run sbin/start-all.sh to start all the daemons. Check if daemon are started using ‘jps’ command.

Analysis 1 : Number of each type of property available in Airbnb Rentals (map, reduce)

- To create database:

  Use airbnb;

- To create collection:

  db.createCollection('listings');

- Import data from local to database by using mongoImport:

  bin/mongoimport --type csv --db airbnb --collection listings --headerline --file /Users/urvashijain/Dataset/listings.csv

Map method :

```
var map = function(){
                emit(this.property_type,1);
        }
```

Reducer method :

```
var reduce  = function(key, value){
                var count = 0;
                for(var i = 0; i < value.length; i++){
                        count += value[i];
                }
                return count;
                }
```

Mapreduce command :

db.listings.mapreduce(map, reduce, {out:"analysis1"});

Output:

```
> map;
function (){
emit(this.property_type,1);
        }
> reduce;
function (key, value){
var count = 0;
for(var i = 0; i < value.length; i++){
count += value[i];
}
return count;
}
> db.listings.mapReduce(map, reduce, {out:"analysis1"});
{
        "result" : "analysis1",
        "timeMillis" : 93,
        "counts" : {
                "input" : 3440,
                "emit" : 3440,
                "reduce" : 203,
                "output" : 21
        },
        "ok" : 1
}
> db.analysis1.find();
{ "_id" : "Aparthotel", "value" : 1 }
{ "_id" : "Apartment", "value" : 2078 }
{ "_id" : "Barn", "value" : 1 }
{ "_id" : "Bed and breakfast", "value" : 83 }
{ "_id" : "Boat", "value" : 3 }
{ "_id" : "Boutique hotel", "value" : 18 }
{ "_id" : "Bungalow", "value" : 3 }
{ "_id" : "Castle", "value" : 1 }
{ "_id" : "Condominium", "value" : 320 }
{ "_id" : "Cottage", "value" : 1 }
{ "_id" : "Guest suite", "value" : 68 }
{ "_id" : "Guesthouse", "value" : 6 }
{ "_id" : "Hostel", "value" : 1 }
{ "_id" : "Hotel", "value" : 22 }
{ "_id" : "House", "value" : 597 }
{ "_id" : "Houseboat", "value" : 2 }
{ "_id" : "Loft", "value" : 30 }
{ "_id" : "Other", "value" : 21 }
{ "_id" : "Serviced apartment", "value" : 90 }
{ "_id" : "Townhouse", "value" : 90 }
Type "it" for more
> it
{ "_id" : "Villa", "value" : 4 }
>
```

Analysis 2 : Calculate average Response rate of each host (map, reduce, finalize)

A host can have multiple hostings(rentals). Thus, the response rate can vary from one rental to another. So, to get the response rate of each host average of all response rates for a particular host is calculated. Finalizer method is used in addition to map and reduce to calculate the average.

Map method:

```
var map = function(){
var response_rate =
parseInt(this.host_response_rate.substring(0,this.host_response_rate.length-1), 10);
if(this.host_name != ''){
if(isNaN(response_rate)){
emit({hostName : this.host_name} , {responseRate : 0 , count : 1});
}else {
emit({hostName : this.host_name} , {responseRate : response_rate  , count : 1});
} } } ;
```

Reducer method:

```
var reduce  = function(key, value){
                var count = {sum:0, totalCount:0};
                for(var i = 0; i < value.length; i++){
                        count.sum += value[i].responseRate;
                        count.totalCount += value[i].count;
                }
                return count;
                };
```

Finalize method:

```
var finalizer = function(key, value){
                if(isNaN(value.sum) || isNaN(value.totalCount)){
                        value.avg = value.responseRate/value.count;
                }else{
                        value.avg = value.sum/value.totalCount;
                }
                return value;
                };
```

Mapreduce command :

db.listings.mapreduce(map, reduce, {out : "analysis2", finalize : finalizer});

Output:

```
> map;
function (){
var response_rate = parseInt(this.host_response_rate.substring(0,this.host_response_rate.length-1), 10);
if(this.host_name != ''){
if(isNaN(response_rate)){
emit({hostName : this.host_name} , {responseRate : 0 , count : 1});
}else {
emit({hostName : this.host_name} , {responseRate : response_rate  , count : 1});
} }
> reduce;
function (key, value){
var count = {sum:0, totalCount:0};
for(var i = 0; i < value.length; i++){
count.sum += value[i].responseRate;
count.totalCount += value[i].count;
}
return count;
}
> finalizer;
function (key, value){
if(isNaN(value.sum) || isNaN(value.totalCount)){
value.avg = value.responseRate/value.count;
}else{
value.avg = value.sum/value.totalCount;
}
return value;
}
> db.listings.mapReduce(map, reduce, {out:"analysis2", finalize:finalizer});
{
        "result" : "analysis2",
        "timeMillis" : 171,
        "counts" : {
                "input" : 3440,
                "emit" : 3436,
                "reduce" : 426,
                "output" : 883
        },
        "ok" : 1
}
> db.analysis2.find();
{ "_id" : { "hostName" : "A And L" }, "value" : { "sum" : 300, "totalCount" : 3, "avg" : 100 } }
{ "_id" : { "hostName" : "A Delightful Hotel" }, "value" : { "sum" : 188, "totalCount" : 2, "avg" : 94 } }
{ "_id" : { "hostName" : "AAA Homestays" }, "value" : { "sum" : 0, "totalCount" : 3, "avg" : 0 } }
{ "_id" : { "hostName" : "Aanchal" }, "value" : { "responseRate" : 0, "count" : 1, "avg" : 0 } }
{ "_id" : { "hostName" : "Aaron" }, "value" : { "sum" : 500, "totalCount" : 5, "avg" : 100 } }
{ "_id" : { "hostName" : "Aaron & Catherine" }, "value" : { "sum" : 200, "totalCount" : 2, "avg" : 100 } }
{ "_id" : { "hostName" : "Abhishek" }, "value" : { "responseRate" : 0, "count" : 1, "avg" : 0 } }
{ "_id" : { "hostName" : "Abigail" }, "value" : { "responseRate" : 100, "count" : 1, "avg" : 100 } }
{ "_id" : { "hostName" : "Acacia" }, "value" : { "responseRate" : 0, "count" : 1, "avg" : 0 } }
{ "_id" : { "hostName" : "Adam" }, "value" : { "sum" : 300, "totalCount" : 4, "avg" : 75 } }
{ "_id" : { "hostName" : "Adolph" }, "value" : { "sum" : 200, "totalCount" : 2, "avg" : 100 } }
{ "_id" : { "hostName" : "Adriana & Gianni" }, "value" : { "sum" : 160, "totalCount" : 2, "avg" : 80 } }
{ "_id" : { "hostName" : "Agathe & Morgan" }, "value" : { "responseRate" : 0, "count" : 1, "avg" : 0 } }
{ "_id" : { "hostName" : "Agnieszka" }, "value" : { "sum" : 0, "totalCount" : 8, "avg" : 0 } }
{ "_id" : { "hostName" : "Aiden & Dana" }, "value" : { "sum" : 300, "totalCount" : 3, "avg" : 100 } }
{ "_id" : { "hostName" : "Aihua" }, "value" : { "sum" : 300, "totalCount" : 3, "avg" : 100 } }
{ "_id" : { "hostName" : "Airbnb" }, "value" : { "sum" : 1673, "totalCount" : 17, "avg" : 98.41176470588235 } }
{ "_id" : { "hostName" : "Ajith Kumar" }, "value" : { "responseRate" : 0, "count" : 1, "avg" : 0 } }
{ "_id" : { "hostName" : "Ak" }, "value" : { "responseRate" : 0, "count" : 1, "avg" : 0 } }
{ "_id" : { "hostName" : "Al" }, "value" : { "responseRate" : 100, "count" : 1, "avg" : 100 } }
Type "it" for more
```

Analysis 3 : Total Number of reviews received by each listing over the years (Composite Key)

The analysis was done to use the concept of composite key to calculate the total number of reviews received by each listing every year from 2009 to 2020 and sort by listing id and year both.

Commands used:

- Create directory in HDFS to store the data

  bin/hadoop fs -mkdir /dataset
- To copy reviews.csv file from local to HDFS directory

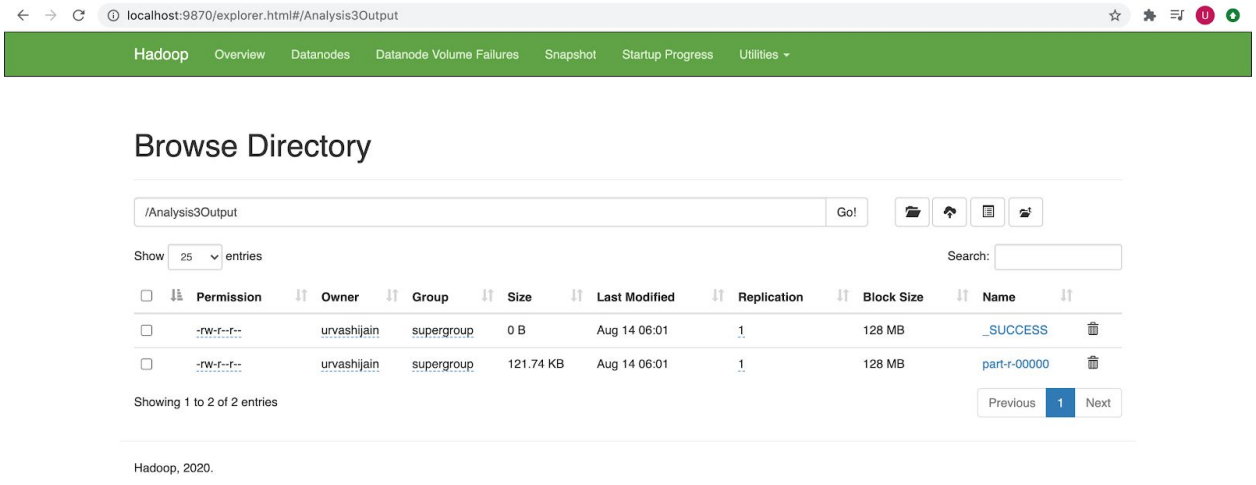
  bin/hadoop fs -copyFromLocal /Users/urvashijain/Dataset/reviews.csv /dataset
- Command to run jar file

  bin/hadoop jar

  /Users/urvashijain/NetBeansProjects/Analysis3/target/Analysis3-1.0-SNAPSHOT.jar

  com.neu.DriverClass /dataset/reviews.csv /Analysis3Output/

Output:



```
3781    2015    5
3781    2016    4
3781    2017    4
3781    2018    1
3781    2019    2
5506    2009    4
5506    2010    10
5506    2011    10
5506    2012    3
5506    2014    3
5506    2015    3
5506    2016    5
5506    2017    12
5506    2018    30
5506    2019    26
5506    2020    1
6695    2009    4
6695    2010    4
6695    2011    4
6695    2012    7
6695    2013    8
6695    2014    14
6695    2015    2
6695    2016    9
6695    2017    13
6695    2018    20
6695    2019    30
8789    2014    5
8789    2015    6
8789    2016    3
8789    2018    8
8789    2019    2
8789    2020    1
10730   2009    5
10730   2010    3
10730   2013    1
10730   2014    5
10730   2015    7
10730   2017    6
10730   2019    4
10730   2020    1
10813   2017    7
10813   2018    38
10813   2019    28
10813   2020    7
10986   2016    2
18711   2010    4
18711   2011    4
18711   2012    4
18711   2014    2
18711   2015    3
18711   2017    15
18711   2018    11
18711   2019    9
22195   2010    2
22195   2013    3
22195   2014    4
22195   2015    1
22195   2019    4
22195   2020    7
22354   2011    14
22354   2012    20
22354   2013    38
22354   2014    74
22354   2015    52
22354   2016    36
22354   2017    27
22354   2018    32
22354   2019    23
```

Observations:

While using composite key, if the data is sorted according to natural key only then it loses a different secondary key with the same natural key. But if a nested condition is used to sort the secondary key then it will give results sorted according to the composite key as a whole.

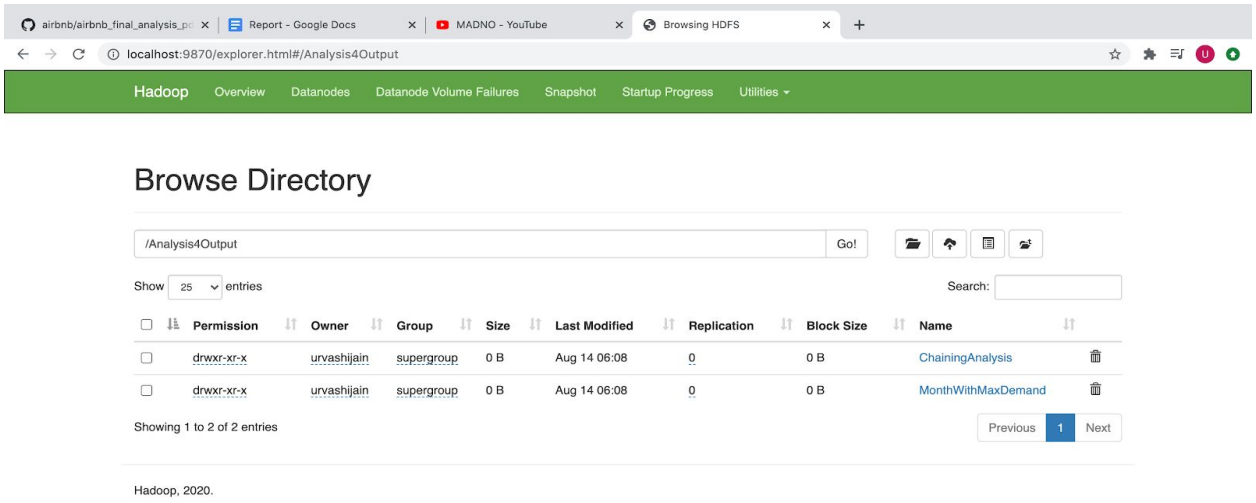
Analysis 4 : Chaining mapreduce to find correlation between demand and price

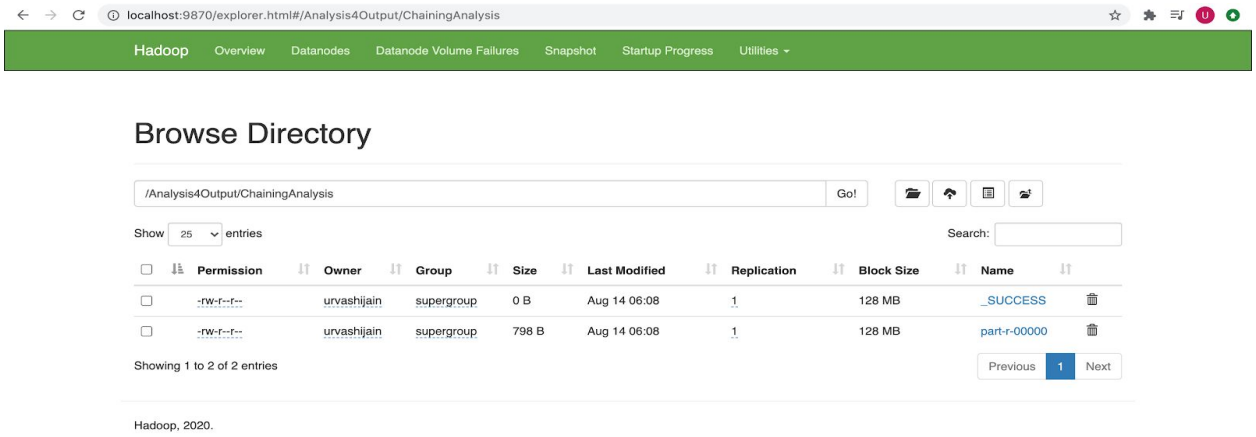
1. **Partitioning data**
   The bookings data of years 2019 and 2020 are divided into 12 partitions, each partition containing data related to a particular month from both the years, utilizing the partitioning pattern of Data Organization Patterns.
2. **Calculate Average**
   The data partitioned in previous analysis is used to calculate the average price of listings and total number of bookings for each month
3. **Sorting**
   The result of the previous step is sorted in descending order, on the basis of the number of bookings to get the month with maximum demand.

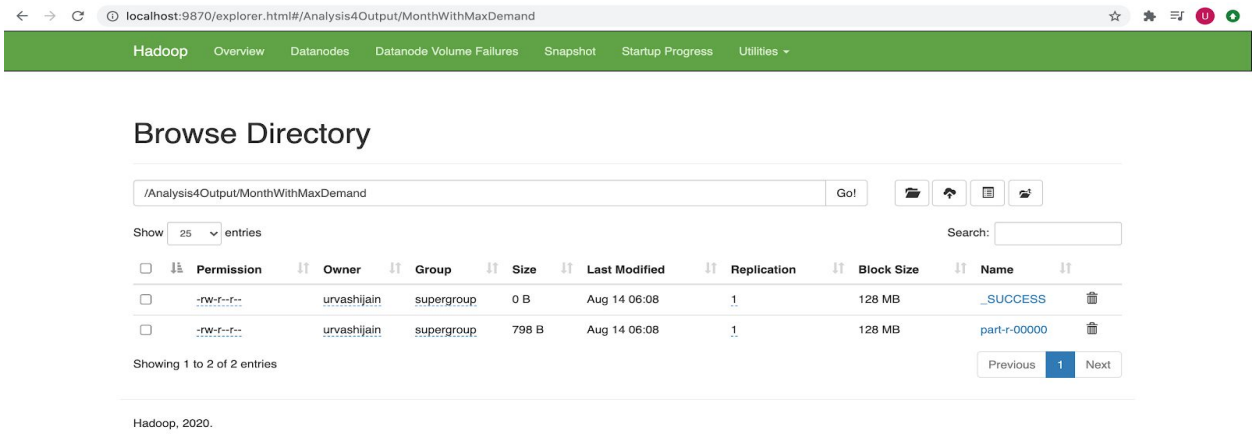
Commands used:

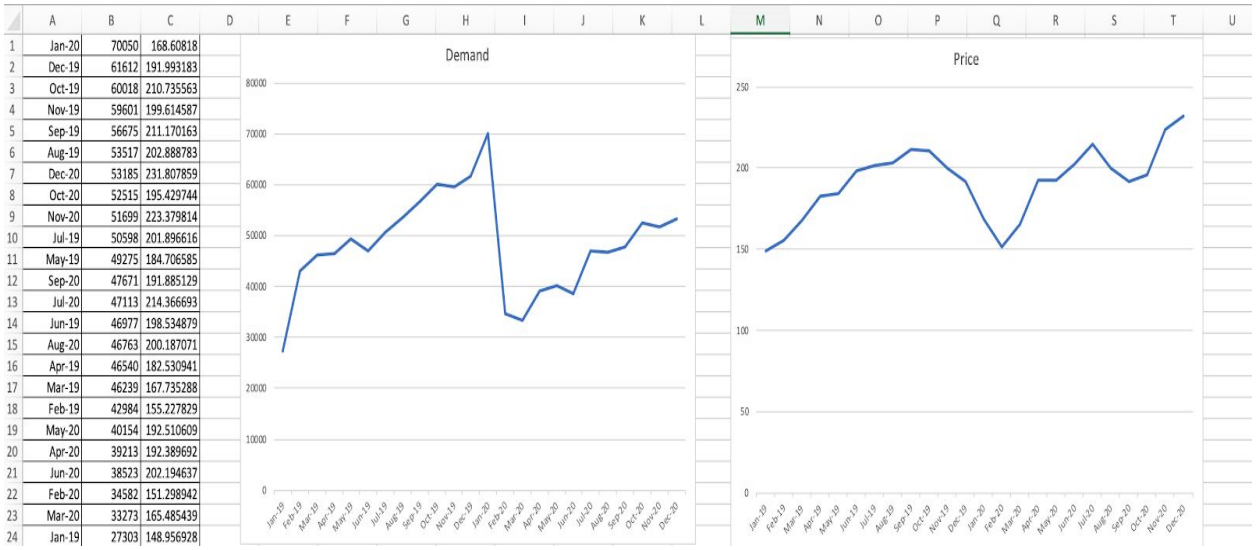
- To copy calendar_2019.csv and calendar_2020.csv files from local to HDFS directory
  bin/hadoop fs -copyFromLocal /Users/urvashijain/Dataset/calendar.csv /dataset*
- Command to run jar file:
  bin/hadoop jar
  /Users/urvashijain/NetBeansProjects/Analysis4/target/Analysis4-1.0-SNAPSHOT.jar
  com.neu.DriverClass /dataset/calendars /Analysis4Output/ChainingAnalysis
  /Analysis4Output/MonthWithMaxDemand

Output:









1	Jan-20	70050	168.60818
2	Dec-19	61612	191.993183
3	Oct-19	60018	210.735563
4	Nov-19	59601	199.614587
5	Sep-19	56675	211.170163
6	Aug-19	53517	202.888783
7	Dec-20	53185	231.807859
8	Oct-20	52515	195.429744
9	Nov-20	51699	223.379814
10	Jul-19	50598	201.896616
11	May-19	49275	184.706585
12	Sep-20	47671	191.885129
13	Jul-20	47113	214.366693
14	Jun-19	46977	198.534879
15	Aug-20	46763	200.187071
16	Apr-19	46540	182.530941
17	Mar-19	46239	167.735288
18	Feb-19	42984	155.227829
19	May-20	40154	192.510609
20	Apr-20	39213	192.389692
21	Jun-20	38523	202.194637
22	Feb-20	34582	151.298942
23	Mar-20	33273	165.485439
24	Jan-19	27303	148.956928

Observation:

The analysis shows the demand increases as the year progresses from Jan to Dec, similar is the case with price. But as we can see from the data, it is not always the case that the maximum demanded month has the maximum average price.

Analysis 5 : First, last and total number of reviews for each listing (Summarization)

Used Summarization Pattern to get the first, last and total number of reviews for each listing.

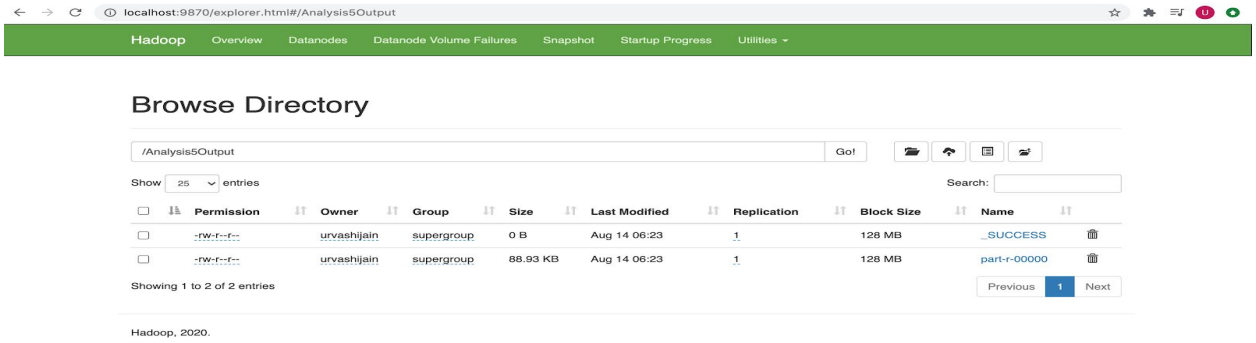
Command:

bin/hadoop jar

/Users/urvashijain/NetBeansProjects/Analysis5/target/Analysis5-1.0-SNAPSHOT.jar

com.neu.DriverClass /dataset/reviews.csv /Analysis5Output

Output:



	Listing ID	First Review	Last Review	Total Reviews
2	3781	7/10/15	12/21/19	16
3	5506	3/21/09	5/1/20	107
4	6695	8/6/09	11/2/19	115
5	8789	8/12/14	4/15/20	25
6	10730	9/21/09	4/16/20	32
7	10813	11/21/17	3/8/20	80
8	10986	5/23/16	5/23/16	2
9	18711	5/12/10	10/5/19	52
10	22195	5/21/10	2/8/20	21
11	22354	8/2/11	12/13/19	316
12	29765	5/24/10	10/21/18	93
13	40601	8/27/10	5/18/20	76
14	45987	8/23/10	12/10/19	130
15	60029	1/1/11	8/13/19	141
16	60356	11/10/13	11/5/18	18
17	67774	11/16/12	10/30/19	44
18	69369	5/23/11	11/15/19	120
19	77681	9/19/11	11/2/19	54
20	95453	4/20/11	10/13/19	41
21	163941	8/19/11	12/30/19	228
22	169430	8/19/11	2/23/20	149
23	182613	8/6/11	6/13/19	106
24	190170	8/15/11	4/2/20	69
25	197727	9/24/11	3/17/20	44
26	197972	10/18/11	3/31/20	388
27	210097	9/16/11	5/31/20	193
28	220676	10/1/11	3/3/20	251
29	225224	10/16/11	3/21/20	318
30	257588	7/20/12	2/17/20	326
31	276450	5/9/12	10/28/19	92
32	295285	5/21/13	3/23/20	30
33	322593	3/27/13	3/31/20	408
34	349347	5/17/12	9/19/19	111

Analysis 6: Top10 and Bottom10 Neighbourhoods (or Boston localities) according to location review score. (Top-N Algorithm)

Applied Top-N Algorithm to find top and bottom 10 localities to the dataset created by calculating average Location review score for each locality in boston.

Commands used:

- To copy listings.csv file from local to HDFS directory

  bin/hadoop fs -copyFromLocal /Users/urvashijain/Dataset/listings.csv /dataset

- Command to run jar file:

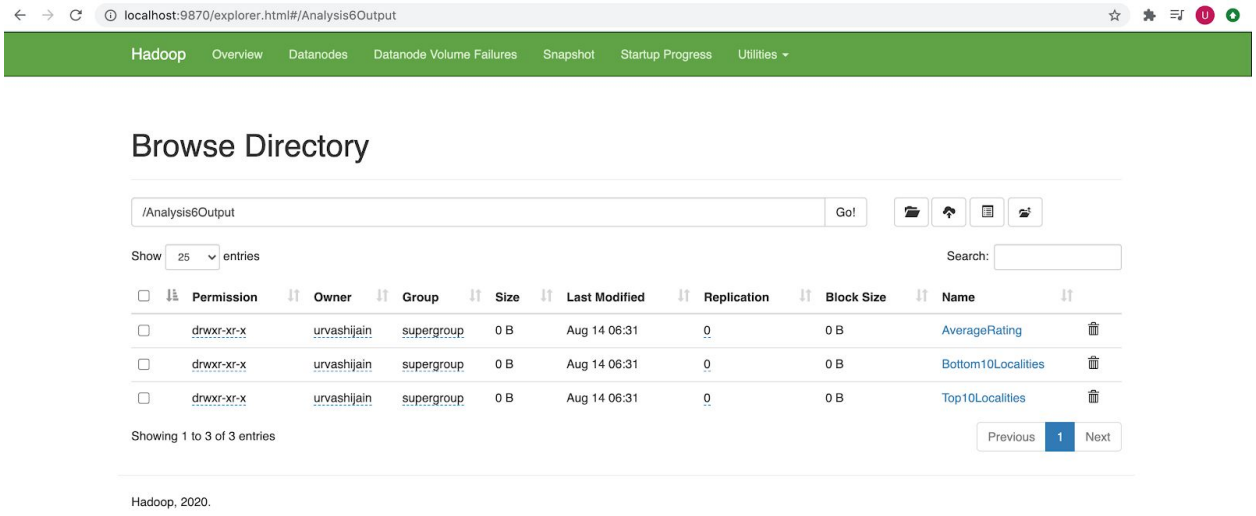
  bin/hadoop jar

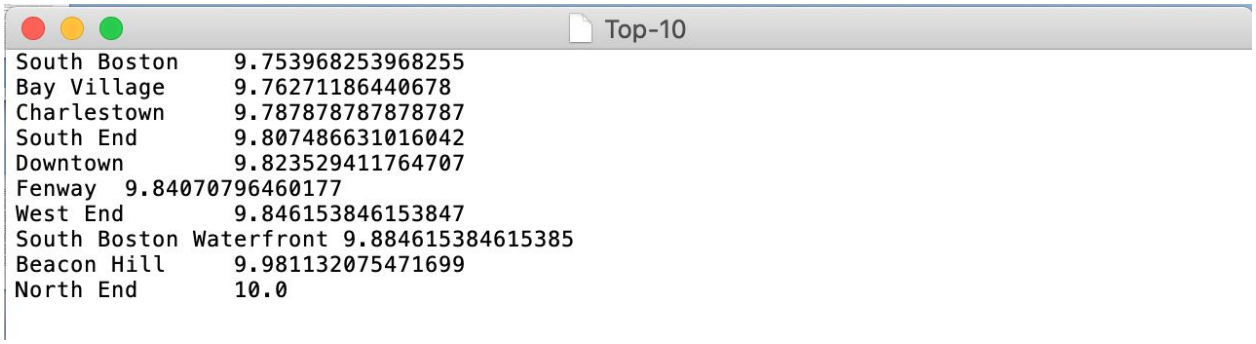
  /Users/urvashijain/NetBeansProjects/Analysis6/target/Analysis6-1.0-SNAPSHOT.jar

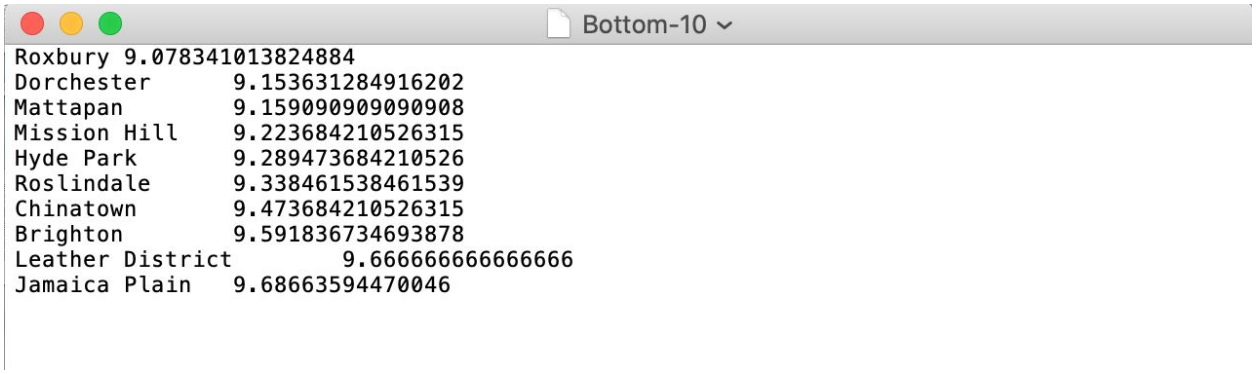
  com.neu.DriverClass /dataset/listings.csv /Analysis6Output/AverageRating

  /Analysis6Output/Top10Localities /Analysis6Output/Bottom10Localities

Output:







Observations:

The Average location review varies between 9 and 10. The Best neighbourhood has highest location review score of 10 and neighbourhood with least score has 9.07 location review score.

Analysis 7: Frequency of booking of rentals depending on verification status, response rate and superhost status of hosts(Join between Calendar and Listings data)

Commands used:

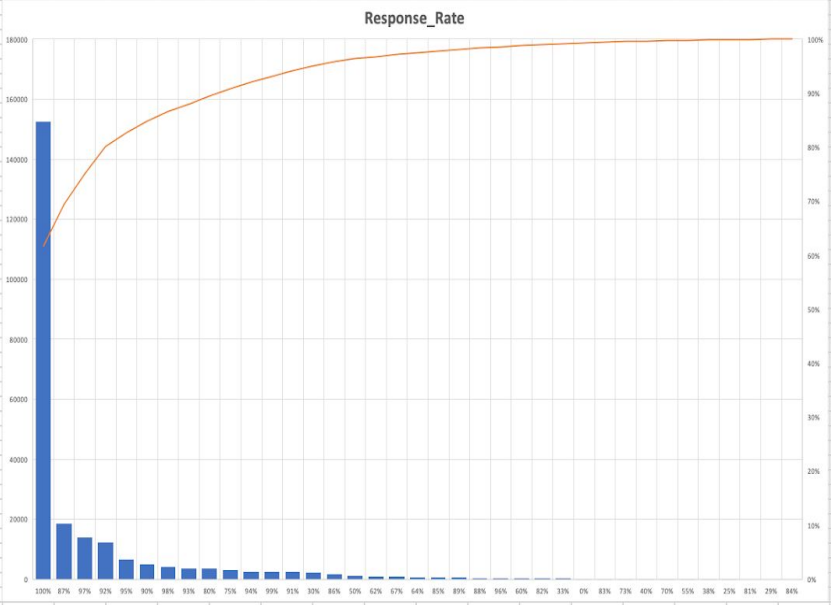
bin/hadoop jar

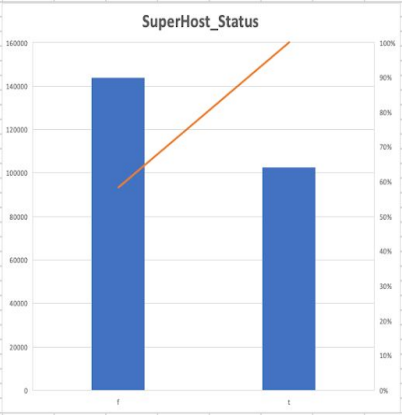
/Users/urvashijain/NetBeansProjects/Analysis7/target/Analysis7-1.0-SNAPSHOT.jar

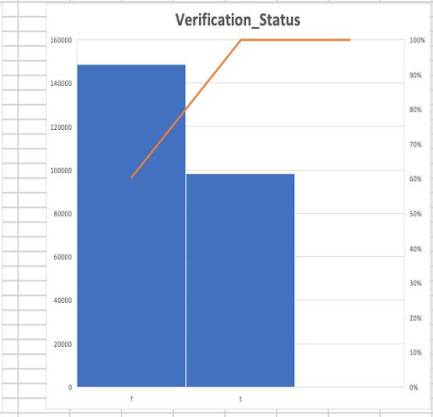
com.neu.DriverClass /dataset/listings.csv /dataset/calendar /Analysis8Output inner

Output:

Superhost	Response rate	verification_Status	Nmber Of Bookings
f	100%	f	44953
f	100%	t	24503
t	100%	f	38690
t	100%	t	44515
f	99%	f	2644
t	98%	t	4210
f	97%	f	4457
f	97%	t	9290
t	97%	t	304
t	96%	f	332
t	96%	t	265
f	95%	f	3594
f	95%	t	1700
t	95%	f	1254
t	95%	t	87
f	94%	f	551
f	94%	t	1741
t	94%	f	289
t	94%	t	198
f	93%	f	3681
f	92%	f	11523
t	92%	f	838
f	91%	f	91
f	91%	t	2180
t	91%	f	324
f	90%	f	1734
f	90%	t	470
t	90%	f	1333
t	90%	t	1591
f	89%	t	390
t	89%	f	290
f	88%	f	287
f	88%	t	167
t	88%	t	177
f	87%	f	18813
f	86%	f	317
t	86%	f	1590
f	85%	f	718
f	84%	f	2
f	83%	f	304
t	83%	f	47
t	83%	t	9
f	82%	f	414
f	81%	t	93
f	80%	f	233
f	80%	t	730
t	80%	f	298
t	80%	t	2404
f	75%	f	2218
f	75%	t	273
t	75%	f	690
t	75%	t	155
t	73%	t	321
t	70%	f	232
f	67%	f	431
t	67%	t	615
t	64%	f	723
f	62%	f	1073
t	60%	t	523
f	55%	t	185
f	50%	f	257
f	50%	t	582
t	50%	f	187
t	50%	t	275
f	40%	f	266
f	38%	f	163
f	33%	f	281
f	33%	t	122
f	30%	f	2298
f	30%	t	67
f	29%	t	10
t	25%	f	119
f	0%	t	365







Observation:

From the above graphs, we can deduce that the number of bookings are highly affected by the host response rate and the superhost status or verification status doesn't play much role in getting higher number of bookings.

Analysis 8: Relation of property types and cancellation policies (Hive Query)

Commands:

1. Create database : *CREATE DATABASE 'airbnb';*
2. Point to that database : *USE airbnb;*
3. Create table :

   Create table listings (id INT, listing_url STRING, last_scraped STRING, picture_url STRING, host_id INT, host_url STRING, host_name STRING, host_since STRING, host_location STRING, host_response_time STRING, host_response_rate INT, host_acceptance_rate INT, host_is_superhost STRING, host_thumbnail_url STRING, host_picture_url STRING, host_neighbourhood STRING, host_listings_count INT, host_total_listings_count INT, host_verifications STRING, host_has_profile_pic STRING, host_identity_verified STRING, street STRING, neighbourhood STRING, neighbourhood_cleansed STRING, city STRING, state STRING, zipcode INT, market STRING, smart_location STRING, country_code STRING, country STRING, latitude FLOAT, longitude FLOAT, is_location_exact STRING, property_type STRING, room_type STRING, accommodates INT, bathrooms INT, bedrooms INT, beds INT, bed_type STRING, amenities STRING, square_feet INT, price STRING, weekly_price STRING, monthly_price STRING, security_deposit STRING, cleaning_fee STRING, guests_included INT, extra_people INT, minimum_nights INT, maximum_nights INT, calendar_updated STRING, has_availability STRING, availability_30 INT, availability_60 INT, availability_90 INT, availability_365 INT, calendar_last_updated STRING, number_of_reviews INT, number_of_reviews_ltm INT, first_review STRING, last_review STRING, review_scores_rating INT, review_scores_accuracy INT, review_scores_cleanliness INT, review_scores_checkin INT, review_scores_communication INT, review_scores_location INT, review_scores_value INT, requires_license STRING, license STRING, jurisdiction_names STRING, instant_bookable STRING, is_business_travel_ready STRING, cancellation_policy STRING, require_guest_profile_picture STRING, require_guest_phone_verification STRING, calculated_host_listings_count INT, calculated_host_listings_count_entire_home INT, calculated_host_listings_count_private_room INT, calculated_host_listings_count_shared_room INT, reviews_per_month FLOAT) ROW FORMAT delimited fields terminated by ",";
4. Load data into the table :

   Load data inpath '/dataset/listings.csv' overwrite into table listings;
5. Query:

   Insert overwrite directory '/Analysis9' row format delimited fields terminated by "\t" select property_type, cancellation_policy, count(*) as count from listings where instant_bookable!='NULL' and cancellation_policy!='NULL' group by property_type, cancellation_policy sort by property_type, count DESC;

6. Output:

	A	B	C
1	property_type	cancellation_policy	Count
2	Aparthotel	flexible	1
3	Apartment	strict_14_with_grace_period	1057
4	Apartment	flexible	496
5	Apartment	moderate	470
6	Apartment	strict	27
7	Apartment	super_strict_30	20
8	Apartment	super_strict_60	7
9	Barn	moderate	1
10	Bed and breakfast	strict_14_with_grace_period	45
11	Bed and breakfast	flexible	21
12	Bed and breakfast	moderate	17
13	Boat	strict_14_with_grace_period	2
14	Boat	moderate	1
15	Boutique hotel	flexible	15
16	Boutique hotel	moderate	2
17	Boutique hotel	strict_14_with_grace_period	1
18	Bungalow	flexible	3
19	Castle	moderate	1
20	Condominium	strict_14_with_grace_period	141
21	Condominium	flexible	89
22	Condominium	moderate	85
23	Condominium	super_strict_30	5
24	Cottage	flexible	1
25	Guest suite	moderate	35
26	Guest suite	strict_14_with_grace_period	18
27	Guest suite	flexible	15
28	Guesthouse	moderate	3
29	Guesthouse	strict_14_with_grace_period	2
30	Guesthouse	flexible	1
31	Hostel	moderate	1
32	Hotel	flexible	22
33	House	strict_14_with_grace_period	287
34	House	moderate	179
35	House	flexible	122
36	House	super_strict_30	6
37	Houseboat	flexible	1
38	Houseboat	strict_14_with_grace_period	1
39	Loft	strict_14_with_grace_period	19
40	Loft	moderate	11
41	Other	moderate	19
42	Other	flexible	1
43	Other	strict_14_with_grace_period	1
44	Serviced apartment	strict_14_with_grace_period	38
45	Serviced apartment	flexible	34
46	Serviced apartment	moderate	18
47	Townhouse	strict_14_with_grace_period	43
48	Townhouse	moderate	25
49	Townhouse	flexible	22
50	Villa	flexible	2
51	Villa	moderate	1
52	Villa	strict_14_with_grace_period	1

Observation:

From the result, we can see the property types like Apartments, Houses, Condominium, Loft, Service Apartments and, Townhouses have strict cancellation policy whereas on the other hand the properties like hotels, Guest suite, etc have moderate to flexible cancellation policies. This implies that the hosts with the whole property as rentals have strict cancellation policies in order to lessen the loss incurred due to last minute cancellations, which is not the case with other types of properties as they have a part of their property on rent. Thus, will not face huge loss due to one cancellation.

Analysis 9: Impact of cleanliness score on bookings in 2020 due to Covid-19 (Pig) (Join between calendar and listings data)

In these unprecedented times of COVID-19, cleanliness has become a major ajenda. Thus, this analysis is done to compare the number of bookings in listings with higher cleanliness score in 2019 and 2020.

Commands:

- To load listings.csv into Pig schema:

  listings = load '/dataset/listings.csv' using PigStorage(',') as (listingId, listing_url, last_scraped, picture_url , host_id, host_url , host_name , host_since , host_location , host_response_time , host_response_rate, host_acceptance_rate, host_is_superhost, host_thumbnail_url , host_picture_url , host_neighbourhood , host_listings_count, host_total_listings_count, host_verifications, host_has_profile_pic, host_identity_verified, street, neighbourhood, neighbourhood_cleansed, city, state, zipcode, market, smart_location, country_code, country, latitude, longitude, is_location_exact, property_type, room_type, accommodates, bathrooms, bedrooms, beds, bed_type, amenities, square_feet, price, weekly_price, monthly_price, security_deposit, cleaning_fee, guests_included, extra_people, minimum_nights, maximum_nights, calendar_updated, has_availability, availability_30, availability_60, availability_90, availability_365, calendar_last_updated, number_of_reviews, number_of_reviews_ltm, first_review, last_review, review_scores_rating, review_scores_accuracy, review_scores_cleanliness, review_scores_checkin, review_scores_communication, review_scores_location, review_scores_value, requires_license, license, jurisdiction_names, instant_bookable, is_business_travel_ready, cancellation_policy, require_guest_profile_picture, require_guest_phone_verification, calculated_host_listings_count, calculated_host_listings_count_entire_home, calculated_host_listings_count_private_room, calculated_host_listings_count_shared_room, reviews_per_month);

- Generate a new schema with required fields

  cleanlinessScoreById = foreach listings generate listingId, review_scores_cleanliness;

- Command to remove blank spaces

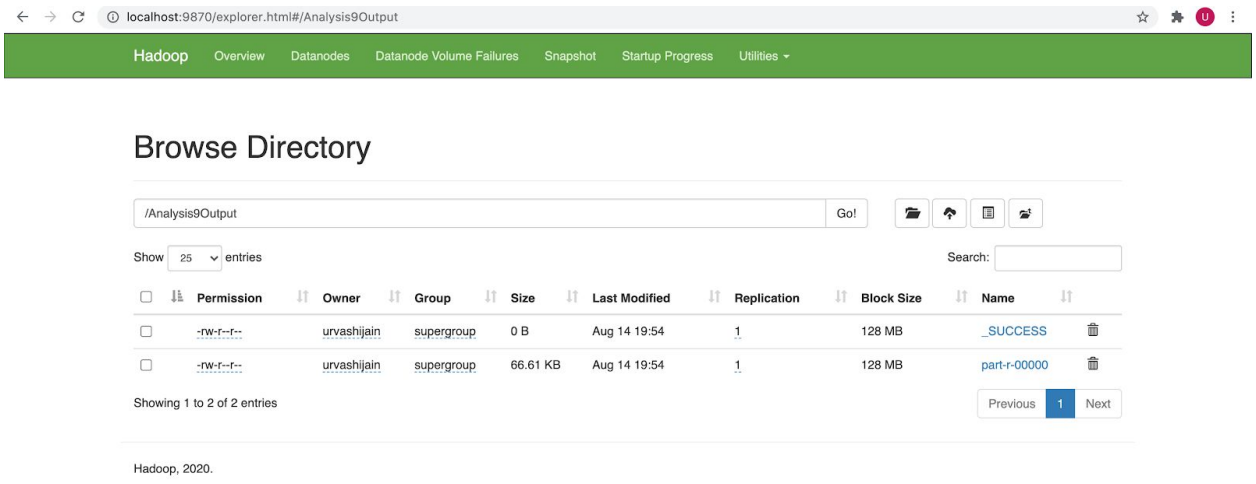
  cleanlinessScoreById = filter cleanlinessScoreById by review_scores_cleanliness>1;

- To load Calendar files in Pig schema from HDFS

  calendar = load '/dataset/calendars' using PigStorage(',') as (listingId, date, available);

- To split date:
  calendar = foreach calendar generate listingId, STRSPLIT(date, '/') as year:(month,date,year), available;
- To get year from date
  calendar = foreach calendar generate listingId, year.year, available;
- Rows with booking as true
  calendar = filter calendar by available=='f';
- Generate alias with required fields
  calendar = foreach calendar generate listingId, year;
- Group on the basis of listings
  bookings = group calendar by (listingId, year);
- Total booking for each listing in respective years
  totalBooking = foreach bookings generate group, COUNT(calendar) as totalBookings;
- Join Operation
  cleanliness_join_bookings = foreach cleanliness_join_bookings generate listingId as id, review_scores_cleanliness as score, group.year as year, totalBookings as bookings;
- Sorting
  sorted_result = order cleanliness_join_bookings by id asc, score desc, year asc ;
- Store the result in a file
  store sorted_result into '/Analysis9Output' using PigStorage();

Output:



	A	B	C	D	E
1	Listings_id	Cleanliness_score	Year	Bookings Count	
2	10034183	10	2019	333	
3	10034183	10	2020	378	
4	10034183	10	2021	3	
5	10068240	10	2019	25	
6	10068240	10	2020	27	
7	10085431	10	2020	332	
8	10085431	10	2021	3	
9	1009275	10	2019	61	
10	1009275	10	2020	44	
11	10118091	10	2019	125	
12	10118091	10	2020	47	
13	10190260	10	2019	261	
14	10190260	10	2020	43	
15	10241577	10	2019	125	
16	10241577	10	2020	1	
17	10249798	10	2019	349	
18	10249798	10	2020	16	
19	10279083	10	2019	171	
20	10279083	10	2020	199	
21	10279083	10	2021	3	
22	10411383	10	2019	17	
23	10411383	10	2020	326	
24	10411383	10	2021	3	
25	10494118	10	2019	233	
26	10494118	10	2020	139	
27	1058448	10	2019	115	
28	1058448	10	2020	4	
29	10634364	10	2019	345	
30	10634364	10	2020	233	
31	10634364	10	2021	3	
32	1071235	10	2019	31	
33	1071235	10	2020	196	
34	1071235	10	2021	3	
35	10749312	10	2019	330	
36	10749312	10	2020	363	
37	10749312	10	2021	3	
38	10764250	10	2019	349	
39	10764250	10	2020	378	
40	10764250	10	2021	3	
41	10787333	10	2019	180	
42	10787333	10	2020	185	
43	10787333	10	2021	3	
44	10813	10	2019	116	
45	10813	10	2020	16	

Observations:

The highlighted rows have more number of bookings in 2020 than 2019 for the same cleanliness score but there are instances when the number of bookings in 2019 are more. Thus, we can conclude that cleanliness is one factor and could be a large factor but the number of bookings do not depend on it entirely.

Analysis 10: How popular has Airbnb become over the years? (Hive)

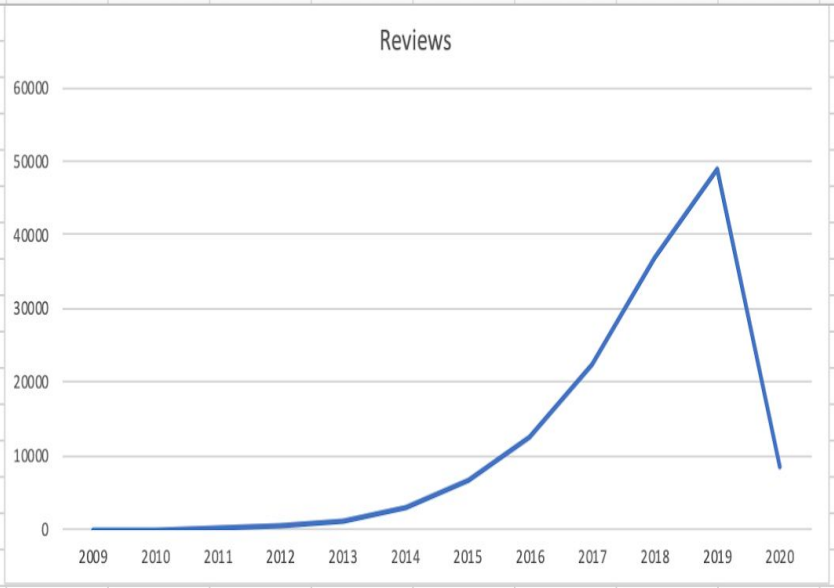
Due to lack of booking data, the number of reviews received over the years is used to study the popularity of airbnb.

Commands:

Insert overwrite directory '/Analysis11' row format delimited fields terminated by "\t" select YEAR(review_date) as year, count(*) from reviews where isNotNull(review_date) group by YEAR(review_date);

Output:

Year	Reviews
2009	13
2010	67
2011	218
2012	469
2013	1056
2014	2891
2015	6736
2016	12434
2017	22575
2018	36993
2019	48956
2020	8484



Observation:

We can infer from the graph that the popularity of airbnb has increased over the years, except for 2020 which is still in progress.

Analysis 11: Mahout Recommendations

Used Mahout Recommendation Engine to provide recommendations to users on the basis of their choices. The engine requires structured data with userId, ItemId and value. The dataset is created using hive.

Commands:

- To create table reviews

  Create table reviews (listing_id INT, review_id INT, review_date DATE, reviewer_id INT, reviewer_name STRING) ROW FORMAT delimited fields terminated by ',';
- To load data from HDFS file to hive table

  Load data inpath '/dataset/reviews.csv' overwrite into table reviews;
- Join Operation on Reviews table and Listings Table

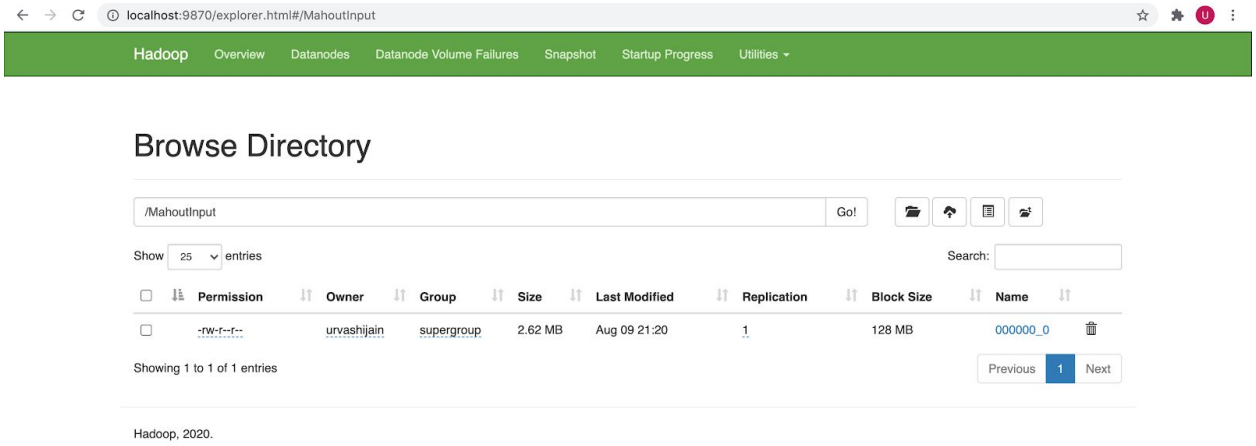
  Insert overwrite directory '/RecommendationAnalysis/' row format delimited fields terminated by "\t" select r.reviewer_id as reviewer_id, l.id as listing_id from listings l join reviews r on (l.id = r.listing_id);
- Create a table mahout_input to provide it as input to Recommendation Engine

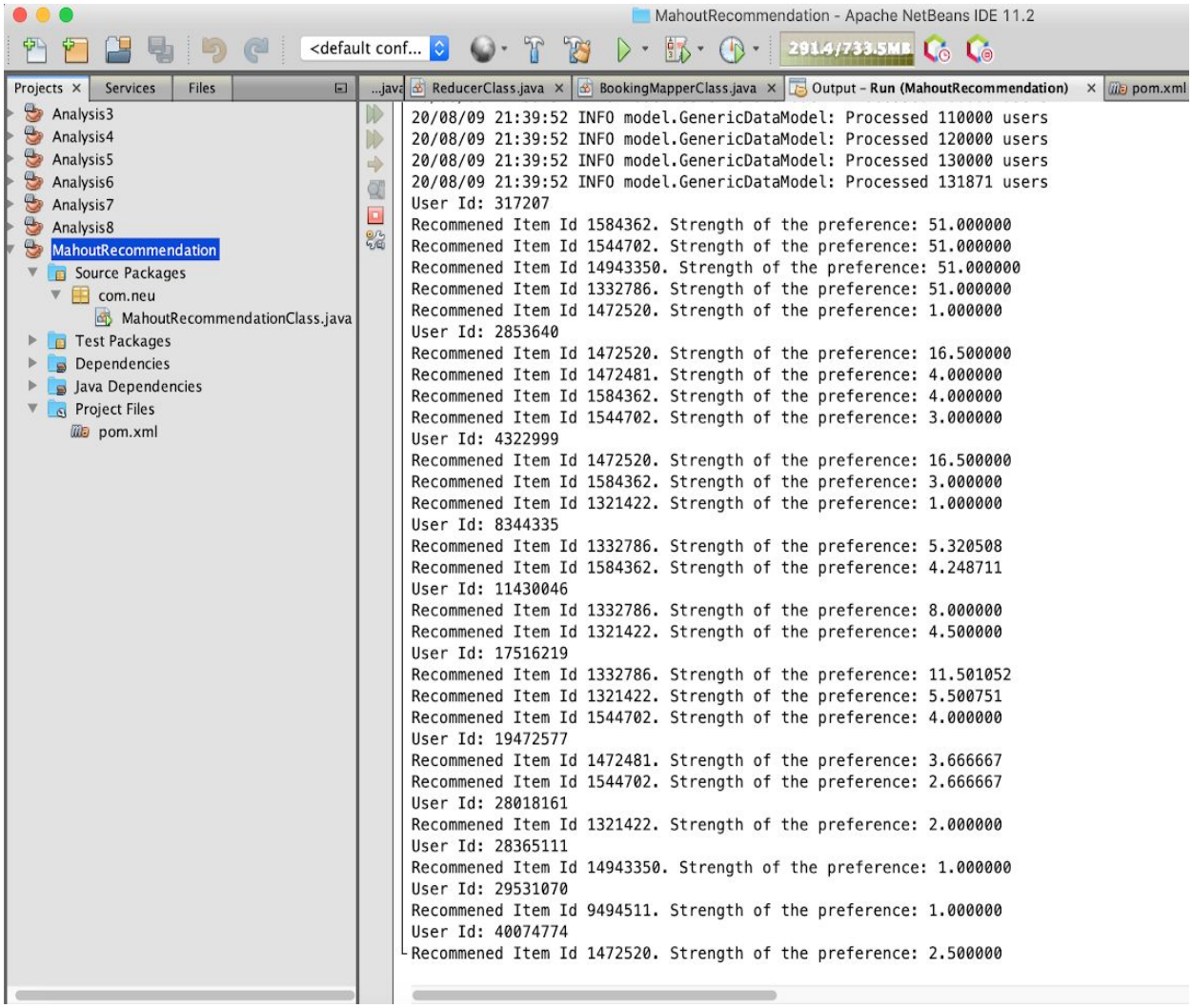
  create table mahout_input (reviewer_id INT, listing_id INT) ROW FORMAT delimited fields terminated by '\t';
- Load data from HDFS to Hive Table

  Load data inpath '/RecommendationAnalysis/' overwrite into table mahout_input;
- Insert the result to HDFS Directory

  Insert overwrite directory '/MahoutInput' row format delimited fields terminated by "\t" select reviewer_id, listing_id, count(*) as rating from mahout_input group by reviewer_id, listing_id;

Output:





Conclusion:

The conclusion from the above analysis is that Airbnb has become popular over the years and the bookings are affected by various factors and do not depend entirely on any one factor. Similarly, hosts have a huge role in affecting the rental bookings.

Appendix

Analysis 3:

Composite Key Class:

```
public class CompositeKeyWritable implements
WritableComparable<CompositeKeyWritable>{
   private long listingId = 0L;
   private int year = 0;

   public CompositeKeyWritable() {
   }

   public CompositeKeyWritable(long listingId, int year) {
      this.listingId = listingId;
      this.year = year;
   }

   public long getListingId() {
      return listingId;
   }

   public void setListingId(long listingId) {
      this.listingId = listingId;
   }

   public int getYear() {
      return year;
   }

    public void setYear(int year) {
      this.year = year;
   }

   @Override
   public void write(DataOutput out) throws IOException {
      out.writeLong(listingId);
      out.writeInt(year);
   }

   @Override
   public void readFields(DataInput in) throws IOException {
      listingId = in.readLong();
      year = in.readInt();
   }
```

```
    @Override
    public int compareTo(CompositeKeyWritable o) {
        int result;
        if(listingId > o.getListingId()){
            result = 1;
        }else if(listingId < o.getListingId()){
            result = -1;
        }else {
            if(year > o.getYear()){
                result = 1;
            }else if(year < o.getYear()){
                result = -1;
            }else{
                result = 0;
            }
        }
        return result;
    }
}
```

Mapper Class:

```
public class MapperClass extends Mapper<LongWritable, Text, CompositeKeyWritable,
IntWritable>{
    private final IntWritable count = new IntWritable(1);
    CompositeKeyWritable obj = new CompositeKeyWritable();

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {

        String line = value.toString();
        String[] tokens = line.split(",");

        if(tokens.length >= 6){
            String year = tokens[2].split("-")[0];
            if(tokens[0].matches("^[0-9]+$") && year.matches("^[0-9]+$")){
                obj.setListingId(Long.parseLong(tokens[0]));
                obj.setYear(Integer.parseInt(year));
                context.write(obj, count);
            }
        }
    }
}
```

Reducer Class:

```
public class ReducerClass extends Reducer<CompositeKeyWritable, IntWritable, Text,
IntWritable>{
   IntWritable value = new IntWritable();
   Text outKey = new Text();

   @Override
   protected void reduce(CompositeKeyWritable key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
      int totalCount = 0;
      for(IntWritable val : values){
         totalCount += val.get();
      }

      value.set(totalCount);
      outKey.set(key.getListingId() + "\t" + key.getYear());
      context.write(outKey, value);
   }
}
```

Driver Class:

```
public class DriverClass {
   public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
      Configuration config = new Configuration();
      Job job = Job.getInstance(config, "Analysis3");
      job.setJarByClass(DriverClass.class);

      job.setMapOutputKeyClass(CompositeKeyWritable.class);
      job.setMapOutputValueClass(IntWritable.class);

      job.setInputFormatClass(TextInputFormat.class);
      job.setOutputFormatClass(TextOutputFormat.class);

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

      job.setMapperClass(MapperClass.class);
      job.setReducerClass(ReducerClass.class);

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

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

Analysis 4:

Composite Key Class:

```
public class CompositeKeyClass implements WritableComparable<CompositeKeyClass>{
   private String month;
   private String numOfBookings;

   public CompositeKeyClass() {
   }

   public CompositeKeyClass(String month, String numOfBookings) {
      this.month = month;
      this.numOfBookings = numOfBookings;
   }

   public String getMonth() {
      return month;
   }

   public void setMonth(String month) {
      this.month = month;
   }

   public String getNumOfBookings() {
      return numOfBookings;
   }

   public void setNumOfBookings(String numOfBookings) {
      this.numOfBookings = numOfBookings;
   }
   @Override
   public void write(DataOutput out) throws IOException {
      out.writeUTF(month);
      out.writeUTF(numOfBookings);
   }

   @Override
   public void readFields(DataInput in) throws IOException {
      month = in.readUTF();
      numOfBookings = in.readUTF();
   }

   @Override
   public int compareTo(CompositeKeyClass obj) {
      int result = this.numOfBookings.compareTo(obj.numOfBookings);
      return (result < 0 ? -1 : (result == 0 ? 0 : 1));
```

```
    }
}
```

Composite Key Comparator:

```
public class CompositeKeyComparator extends WritableComparator {
    public CompositeKeyComparator(){
        super(CompositeKeyClass.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        CompositeKeyClass ckw1 = (CompositeKeyClass) a;
        CompositeKeyClass ckw2 = (CompositeKeyClass) b;
        int result = -1 * ckw1.getNumOfBookings().compareTo(ckw2.getNumOfBookings());
        return result;
    }
}
```

Natural Key Grouping Comparator:

```
public class NaturalKeyGroupingComparator extends WritableComparator {
    public NaturalKeyGroupingComparator(){
        super(CompositeKeyClass.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        CompositeKeyClass ckw1 = (CompositeKeyClass) a;
        CompositeKeyClass ckw2 = (CompositeKeyClass) b;
        int result = ckw1.getMonth().compareTo(ckw2.getMonth());
        return result;
    }
}
```

Natural Key Partitioner:

```
public class NaturalKeyPartitioner extends Partitioner<CompositeKeyClass, IntWritable> {
    @Override
    public int getPartition(CompositeKeyClass key, IntWritable value, int noOfPartitions) {
        return key.getMonth().hashCode() % noOfPartitions;
    }
}
```

Partitioner Mapper Class:

```
public class PartitionMapperClass extends Mapper<LongWritable, Text, Text, Text>{
   private Text outKey = new Text();

   @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
          InterruptedException {
      String line = value.toString();
      String[] tokens = line.split(",");
      String month_year = null;

      if(!tokens[1].equals("date") && tokens[2].equals("f")){
         String[] date = tokens[1].split("/");
         month_year = date[0] + "-20" + date[2];
         outKey.set(month_year);
         context.write(outKey, value);
      }
   }
}
```

Average and count Mapper Class:

```
public class AverageAndCountMapper extends Mapper<Text, Text, Text, Text>{
   Text outVal = new Text();

   @Override
       protected void map(Text key, Text value, Context context) throws IOException,
InterruptedException {

      String line = value.toString();
      String[] tokens = line.split(",");

      if(!tokens[3].isEmpty()){
         outVal.set("1" + " , "+ tokens[3]);
         context.write(key, outVal);
      }
   }
}
```

Sorting Mapper Class:

```
public class SortingMapper extends Mapper<LongWritable, Text, CompositeKeyClass,
DoubleWritable> {
   DoubleWritable outVal = new DoubleWritable();

   @Override
   protected void map(LongWritable key, Text value, Context context)
   throws IOException,   InterruptedException {
      String line = value.toString();
      String[] tokens = line.split("\t");
      if (tokens.length == 3) {
         String numOfBookings = tokens[1].trim();
         outVal.set(Double.parseDouble(tokens[2].trim()));
         CompositeKeyClass obj = new CompositeKeyClass(tokens[0].trim(), numOfBookings);
         context.write(obj, outVal);
      }
   }
}
```

Reducer Class for Chained Mappers:

```
public class ReducerClass extends Reducer<Text, Text, Text, Text>{
   Text outVal = new Text();
   @Override
       protected void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
      int numberOfBookings = 0;
      double totalPrice = 0;
      double avg = 0 ;

      for(Text val : values){
         String[] tokens = val.toString().split(",");
            if(tokens.length == 2){
            numberOfBookings += Integer.parseInt(tokens[0].trim());
            if(!tokens[1].contains("$")){
               totalPrice += Double.parseDouble(tokens[1].trim());
            }
         }
      }

      avg = totalPrice/numberOfBookings;
      String finalval = numberOfBookings + "\t" + avg;
      outVal.set(finalval);
      context.write(key, outVal);
   }
}
```

Sorting Reducer Class:

```
public class SortingReducerClass extends Reducer<CompositeKeyClass, DoubleWritable,
        Text, DoubleWritable>{

   @Override
   protected void reduce(CompositeKeyClass key, Iterable<DoubleWritable> values, Context
         context) throws IOException, InterruptedException {
      for(DoubleWritable val : values){
         Text outKey = new Text();
         outKey.set(key.getMonth() + "\t" + key.getNumOfBookings());
         context.write(outKey, val);
      }
   }
}
```

Driver Class:

```
public class DriverClass {
      public static void main(String[] args) throws IOException, ClassNotFoundException,
         InterruptedException {
         Configuration conf = new Configuration();
         Job job = new Job(conf, "ChainingAnalysis");
         job.setJarByClass(DriverClass.class);

          // MapReduce chaining
         Configuration mapConf1 = new Configuration(false);
               ChainMapper.addMapper(job,  PartitionMapperClass.class,  LongWritable.class,
         Text.class, Text.class, Text.class, mapConf1);

          job.setPartitionerClass(PartitionerClass.class);
         PartitionerClass.setMinLastAccessDate(job, 2019);

          Configuration mapConf2 = new Configuration(false);
         ChainMapper.addMapper(job, AverageAndCountMapper.class, Text.class, Text.class,
         Text.class, Text.class, mapConf2);

          Configuration reduceConf = new Configuration(false);
          ChainReducer.setReducer(job, ReducerClass.class, Text.class, Text.class, Text.class,
         Text.class, reduceConf);

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

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

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

```
        boolean result = job.waitForCompletion(true);
        if(result){
        Job job1= new Job(conf, "SecondarySortAnalysis");
        job1.setJarByClass(DriverClass.class);

        job1.setMapperClass(SortingMapper.class);
        job1.setReducerClass(SortingReducerClass.class);

        job1.setInputFormatClass(TextInputFormat.class);
        job1.setOutputFormatClass(TextOutputFormat.class);

        job1.setMapOutputKeyClass(CompositeKeyClass.class);
        job1.setMapOutputValueClass(DoubleWritable.class);

        job1.setOutputKeyClass(Text.class);
        job1.setOutputValueClass(DoubleWritable.class);

        FileInputFormat.addInputPath(job1, new Path(args[1]));
        FileOutputFormat.setOutputPath(job1, new Path(args[2]));

        job1.setGroupingComparatorClass(NaturalKeyGroupingComparator.class);
        job1.setSortComparatorClass(CompositeKeyComparator.class);
        job1.setPartitionerClass(NaturalKeyPartitioner.class);

        job1.setNumReduceTasks(1);
        result = job1.waitForCompletion(true);
    }
  }
}
```

Analysis 5:

Composite Key:

```
public class CompositeReviewValueClass implements Writable{

   private Date firstReview = new Date();
   private Date lastReview = new Date();
   private int totalreviews = 0;

   private final static SimpleDateFormat SIMPLE_DATE_FORMAT = new
         SimpleDateFormat("yyyy-MM-dd");


   public CompositeReviewValueClass() {
   }

   public CompositeReviewValueClass(Date firstReview, Date lastReview, int totalreviews) {
      this.firstReview = firstReview;
      this.lastReview = lastReview;
      this.totalreviews = totalreviews;
   }

   public Date getFirstReview() {
      return firstReview;
   }

   public void setFirstReview(Date firstReview) {
      this.firstReview = firstReview;
   }

   public Date getLastReview() {
      return lastReview;
   }

   public void setLastReview(Date lastReview) {
      this.lastReview = lastReview;
   }

   public int getTotalreviews() {
      return totalreviews;
   }

   public void setTotalreviews(int totalreviews) {
      this.totalreviews = totalreviews;
   }

   @Override
```

```
    public void write(DataOutput out) throws IOException {
        out.writeLong(firstReview.getTime());
        out.writeLong(lastReview.getTime());
        out.writeInt(totalreviews);
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        firstReview = new Date(in.readLong());
        lastReview = new Date(in.readLong());
        totalreviews = in.readInt();
    }

    @Override
    public String toString(){
        return SIMPLE_DATE_FORMAT.format(firstReview) + "\t" +
            SIMPLE_DATE_FORMAT.format(lastReview) + "\t" + totalreviews;
    }
}
```

Mapper Class:

```
public class MapperClass extends Mapper<LongWritable, Text, IntWritable,
            CompositeReviewValueClass>{
    IntWritable outKey = new IntWritable();
    CompositeReviewValueClass outVal = new CompositeReviewValueClass();

    private final static SimpleDateFormat SIMPLE_DATE_FORMAT = new
            SimpleDateFormat("yyyy-MM-dd");

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
            InterruptedException {
        String line = value.toString();
        String[] tokens = line.split(",");
        try {
            if(tokens.length>3){
                if(!tokens[0].equals("id") && !tokens[2].equals("date")){
                    outVal.setFirstReview(SIMPLE_DATE_FORMAT.parse(tokens[2]));
                    outVal.setLastReview(SIMPLE_DATE_FORMAT.parse(tokens[2]));
                    outVal.setTotalreviews(1);
                    outKey.set(Integer.parseInt(tokens[0]));
                    context.write(outKey, outVal);
                }
            }
        } catch (ParseException ex) {
            System.out.println(ex.getMessage());
        } } }
```

Reducer Class:

```
public class MapperClass extends Mapper<LongWritable, Text, IntWritable,
        CompositeReviewValueClass>{
   IntWritable outKey = new IntWritable();
   CompositeReviewValueClass outVal = new CompositeReviewValueClass();
   private final static SimpleDateFormat SIMPLE_DATE_FORMAT = new
        SimpleDateFormat("yyyy-MM-dd");

   @Override
   protected void map(LongWritable key, Text value, Context context) throws IOException,
        InterruptedException {
      String line = value.toString();
      String[] tokens = line.split(",");
      try {
         if(tokens.length>3){
            if(!tokens[0].equals("id") && !tokens[2].equals("date")){
               outVal.setFirstReview(SIMPLE_DATE_FORMAT.parse(tokens[2]));
               outVal.setLastReview(SIMPLE_DATE_FORMAT.parse(tokens[2]));
               outVal.setTotalreviews(1);
               outKey.set(Integer.parseInt(tokens[0]));
               context.write(outKey, outVal);
            } }
      } catch (ParseException ex) {
         System.out.println(ex.getMessage());
      } } }
```

Driver Class:

```
public class DriverClass {
   public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "Analysis6");
        job.setJarByClass(DriverClass.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(CompositeReviewValueClass.class);
        job.setMapperClass(MapperClass.class);
        job.setReducerClass(ReducerClass.class);
        job.setCombinerClass(Reducer.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(CompositeReviewValueClass.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        System.exit(job.waitForCompletion(true)? 0:1);
        } }
```

Analysis 6:

Average Mapper Class:

```
public class AverageMapperClass extends Mapper<LongWritable, Text, Text, Text>{
   private Text outValue = new Text();
   private Text neighbourhood = new Text();

   @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException,
          InterruptedException{
      String line  = value.toString();
      String[] tokens = line.split(",");
      if(tokens.length == 83){
         if(tokens[68].matches("^[0-9]+$") && !tokens[23].trim().equals("neighbourhood")){
            neighbourhood.set(tokens[23]);
            outValue.set(tokens[68] + "\t"+ "1");
            context.write(neighbourhood, outValue);
         }
      }
   }
}
```

Average Reducer Class:

```
public class AverageReducerClass extends Reducer<Text, Text, Text, DoubleWritable>{
   private DoubleWritable avgRatingScore = new DoubleWritable();

   @Override
       protected void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
      int totalCount = 0;
      double totalReviewScore = 0;

      for(Text val : values){
         String[] tokens = val.toString().split("\t");
         totalCount += Integer.parseInt(tokens[1]);
         totalReviewScore += Integer.parseInt(tokens[0]);
      }
      double avg = (double) totalReviewScore/totalCount;
      avgRatingScore.set(avg);
      context.write(key, avgRatingScore);
   }
}
```

Top 10 Mapper Class:

```
public class Top10MapperClass extends Mapper<Object, Text, Text, DoubleWritable> {
   private TreeMap<Double, String> tmap;

   @Override
   public void setup(Context context) throws IOException, InterruptedException{
      tmap = new TreeMap<Double, String>();
   }

   @Override
   public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
      String[] tokens = value.toString().split("\t");
      String neighbourhood = tokens[0];
      double avgLocationReviewScore = Double.parseDouble(tokens[1]);
      tmap.put(avgLocationReviewScore, neighbourhood);
      if (tmap.size() > 10) {
         tmap.remove(tmap.firstKey());
      }
   }

   @Override
   public void cleanup(Context context) throws IOException, InterruptedException {
      for (Map.Entry<Double, String> entry : tmap.entrySet()){
         double avgRating = entry.getKey();
         String neighbourhood = entry.getValue();
         context.write(new Text(neighbourhood), new DoubleWritable(avgRating));
      }
   }
}
```

Top 10 Reducer Class:

```
public class Top10ReducerClass extends Reducer<Text, DoubleWritable, Text,
DoubleWritable> {
   private TreeMap<Double, String> tmap2;

   @Override
   public void setup(Context context) throws IOException, InterruptedException{
      tmap2 = new TreeMap<Double, String>();
   }

   @Override
   public void reduce(Text key, Iterable<DoubleWritable> values, Context context)
   throws IOException, InterruptedException {
      String name = key.toString();
      double ratingScore = 0;
      for (DoubleWritable val : values) {
         ratingScore = val.get();
      }
      tmap2.put(ratingScore, name);
      if (tmap2.size() > 10){
         tmap2.remove(tmap2.firstKey());
      }
   }

   @Override
   public void cleanup(Context context) throws IOException, InterruptedException{
      for (Map.Entry<Double, String> entry : tmap2.entrySet())  {
         double ratingScore = entry.getKey();
         String neighbourhood = entry.getValue();
         context.write(new Text(neighbourhood), new DoubleWritable(ratingScore));
      }
   }
}
```

Bottom 10 Mapper Class:

```
public class Bottom10MapperClass extends Mapper<LongWritable, Text, Text,
DoubleWritable> {
   private TreeMap<Double, String> tmap;

   @Override
   public void setup(Mapper.Context context) throws IOException, InterruptedException{
      tmap = new TreeMap<Double, String>();
   }

   @Override
   public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      String[] tokens = value.toString().split("\t");
      String neighbourhood = tokens[0];
      double avgRatingScore = Double.parseDouble(tokens[1]);
      tmap.put(avgRatingScore, neighbourhood);

      if (tmap.size() > 10) {
         tmap.remove(tmap.lastKey());
      }
   }

   @Override
   public void cleanup(Mapper.Context context) throws IOException, InterruptedException {
      for (Map.Entry<Double, String> entry : tmap.entrySet()){
         double avgRating = entry.getKey();
         String neighbourhood = entry.getValue();
         context.write(new Text(neighbourhood), new DoubleWritable(avgRating));
      }
   }
}
```

Bottom 10 Reducer Class:

```
public class Bottom10ReducerClass extends Reducer<Text, DoubleWritable, Text,
DoubleWritable> {

   private TreeMap<Double, String> tmap2;

   @Override
   public void setup(Reducer.Context context) throws IOException, InterruptedException{
      tmap2 = new TreeMap<Double, String>();
   }

   @Override
   public void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws
IOException, InterruptedException {
      String name = key.toString();
      double ratingScore = 0;

      for (DoubleWritable val : values) {
         ratingScore = val.get();
      }

      tmap2.put(ratingScore, name);

      if (tmap2.size() > 10){
         tmap2.remove(tmap2.lastKey());
      }
   }

   @Override
   public void cleanup(Reducer.Context context) throws IOException, InterruptedException{
      for (Map.Entry<Double, String> entry : tmap2.entrySet())  {
         double ratingScore = entry.getKey();
         String neighbourhood = entry.getValue();
         context.write(new Text(neighbourhood), new DoubleWritable(ratingScore));
      }
   }
}
```

Driver Class:

```
public class DriverClass {
   public static void main(String[] args)
   throws IOException, ClassNotFoundException, InterruptedException {
          Configuration conf = new Configuration();
          Job job = new Job(conf, "AverageRating");
          Job job1 = new Job(conf, "Top10Neighbourhoods");
          Job job2 = new Job(conf, "Bottom10Neighbourhoods");
```

```
        job.setJarByClass(DriverClass.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        job.setMapperClass(AverageMapperClass.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(DoubleWritable.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

    if(job.waitForCompletion(true)){
       job1.setJarByClass(DriverClass.class);
       FileInputFormat.addInputPath(job1, new Path(args[1]));
       FileOutputFormat.setOutputPath(job1, new Path(args[2]));
       job1.setMapOutputKeyClass(Text.class);
       job1.setMapOutputValueClass(DoubleWritable.class);

       job1.setMapperClass(Top10MapperClass.class);
       job1.setReducerClass(Top10ReducerClass.class);

       job1.setOutputKeyClass(Text.class);
       job1.setOutputValueClass(DoubleWritable.class);

       job1.setInputFormatClass(TextInputFormat.class);
       job1.setOutputFormatClass(TextOutputFormat.class);
    }
    if(job1.waitForCompletion(true)){
       job2.setJarByClass(DriverClass.class);
       FileInputFormat.addInputPath(job2, new Path(args[1]));
       FileOutputFormat.setOutputPath(job2, new Path(args[3]));

       job2.setMapOutputKeyClass(Text.class);
       job2.setMapOutputValueClass(DoubleWritable.class);

       job2.setMapperClass(Bottom10MapperClass.class);
       job2.setReducerClass(Bottom10ReducerClass.class);

       job2.setOutputKeyClass(Text.class);
       job2.setOutputValueClass(DoubleWritable.class);

       job2.setInputFormatClass(TextInputFormat.class);
       job2.setOutputFormatClass(TextOutputFormat.class);
       job2.waitForCompletion(true);
    }
  }
}
```

Analysis 7:

Booking Mapper Class:

```
public class BookingMapperClass extends Mapper<LongWritable, Text, IntWritable, Text>{
   private IntWritable listingId = new IntWritable();
   private Text booking = new Text("B1");
   @Override
   protected void map(LongWritable key, Text value, Mapper.Context context){
      String record = value.toString();
      String[] tokens = record.split(",");
      try {
         if(tokens[0].matches("^[0-9]+$")){
            if(tokens[2].equals("f")){
               listingId.set(Integer.parseInt(tokens[0]));
               context.write(listingId, booking);
            }
         }
      } catch (IOException | InterruptedException ex) {
         System.out.println(ex.getMessage());
      }
   }
}
```

Listing Mapper Class:

```
public class ListingMapperClass extends Mapper<LongWritable, Text, IntWritable, Text>{
   private IntWritable listingId = new IntWritable();
   private Text outVal = new Text();
   @Override
   protected void map(LongWritable key, Text value, Context context){
      String record = value.toString();
      String[] tokens = record.split(",");
      try {
         if(tokens.length >=82){
            if(tokens[0].matches("^[0-9]+$") && !tokens[10].isEmpty() && !tokens[20].isEmpty()
                  && (tokens[12].equals("t") || tokens[12].equals("f")) &&
                  (tokens[20].equals("t") || tokens[20].equals("f"))){
               if(tokens[10].substring(0, tokens[10].length()-1).matches("^[0-9]+$")){
                  listingId.set(Integer.parseInt(tokens[0]));
                  outVal.set("A"+ "\t" + tokens[12] + "\t" + tokens[10]+ "\t"+ tokens[20]);
                  context.write(listingId, outVal);
               }
            }
         }
      } catch (IOException | InterruptedException ex) {
         System.out.println(ex.getMessage());
      }
   } }
```

Mapper Class:

```
public class MapperClass extends Mapper<LongWritable, Text, Text, IntWritable>{

   private IntWritable outVal = new IntWritable();
   private Text outKey = new Text();

   @Override
   protected void map(LongWritable key, Text value, Context context){
      String record = value.toString();
      String[] tokens = record.split("\t");
      try {
         if(tokens.length == 5){
            outKey.set(tokens[1] + "\t" + tokens[2] + "\t" + tokens[3]);
            outVal.set(Integer.parseInt(tokens[4]));
            context.write(outKey, outVal);
         }
      } catch (IOException | InterruptedException ex) {
         System.out.println(ex.getMessage());
      }
   }
}
```

Reducer Class:

```
public class ReducerClass extends Reducer<IntWritable, Text, IntWritable, Text>{
   private List<Text> listA = new ArrayList<Text>();
   private List<Text> listB = new ArrayList<Text>();
   private String joinType = null;

   @Override
   protected void setup(Context context) throws IOException, InterruptedException {
      joinType = context.getConfiguration().get("join.type");
   }

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

      listA.clear();
      listB.clear();
      int bookings = 0;

      for(Text val : values){
         if(val.toString().charAt(0) == 'A'){
            String[] tokens = val.toString().split("\t");
            listA.add(new Text(tokens[1]+"\t"+tokens[2]+"\t"+tokens[3]));
         }else if(val.toString().charAt(0) == 'B'){
            bookings += Integer.parseInt(val.toString().substring(1));
```

```
        }
      }

      listB.add(new Text(String.valueOf(bookings)));

      if(joinType.equals("inner")){
         if(!listA.isEmpty() && !listB.isEmpty()){
            for(Text hostDetails : listA){
               for(Text numOfBookings: listB){
                                    context.write(key,  new  Text(String.valueOf(hostDetails  +  "\t"+
numOfBookings)));
               }
            }
         }
      }
   }
}
```

Final Reducer Class:

```
public class FinalReducerClass extends Reducer<Text, IntWritable, Text, IntWritable>{

   @Override
     protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
      int totalBookings = 0;
      for(IntWritable vals : values){
         totalBookings += vals.get();
      }
      context.write(key, new IntWritable(totalBookings));
   }
}
```

Driver Class:

```
public class DriverClass {
        public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException{
      Configuration conf = new Configuration();
      Job job = new Job(conf, "Analysis8");
      job.setJarByClass(DriverClass.class);

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

         MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, ListingMapperClass.class);
      MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, BookingMapperClass.class);

      FileOutputFormat.setOutputPath(job, new Path(args[2]));
      job.getConfiguration().set("join.type" , args[3]);

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

      job.setReducerClass(ReducerClass.class);
      job.setOutputFormatClass(TextOutputFormat.class);

      if(job.waitForCompletion(true)){
         Job job2 = new Job(conf, "FinalAnalysis");

         FileInputFormat.addInputPath(job2, new Path(args[2]));
         FileOutputFormat.setOutputPath(job2, new Path(args[4]));

         job2.setMapOutputKeyClass(Text.class);
         job2.setMapOutputValueClass(IntWritable.class);

         job2.setMapperClass(MapperClass.class);
         job2.setReducerClass(FinalReducerClass.class);

         job2.setOutputKeyClass(Text.class);
         job2.setOutputValueClass(IntWritable.class);

         job2.setInputFormatClass(TextInputFormat.class);
         job2.setOutputFormatClass(TextOutputFormat.class);
         job2.waitForCompletion(true);
      }
   }
}
```

Analysis 11:

Mahout Recommendation Engine Class:

```
public class MahoutRecommendationClass {

   public static void main(String[] args) {
      try {
                                                DataModel model = new FileDataModel(new
File("/Users/urvashijain/Dataset/MahoutInput.tsv"));
          UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
            UserNeighborhood neighborhood = new NearestNUserNeighborhood(2, similarity,
model);
                Recommender recommender = new GenericUserBasedRecommender(model,
neighborhood, similarity);

          for (LongPrimitiveIterator iterator = model.getUserIDs(); iterator.hasNext();) {
             long userId = iterator.nextLong();
                                                List<RecommendedItem> itemRecommendations =
recommender.recommend(userId, 5);

             //System.out.format("User Id: %d%n", userId);

             if (itemRecommendations.isEmpty()) {
                //System.out.println("No recommendations for this user.");
             } else {
                System.out.format("User Id: %d%n", userId);
                for (RecommendedItem recommendedItem : itemRecommendations) {
                        System.out.format("Recommened Item Id %d. Strength of the preference:
%f%n", recommendedItem.getItemID(), recommendedItem.getValue());
                }
             }
          }
      } catch (IOException | TasteException ex) {
                           System.out.println("Exception in MahoutRecommendationClass: " +
ex.getMessage());
      }
   }
}
```