MongoDB commands

1. Insert records into MongoDB Collection

Code:

```
db.cis436.insert(UserId: '9', ProductId: '500', Ratings:'5');
```

(Here, the insert command inserts new data in the collection.)

2. Update records into MongoDB collection

Code:

```
Db.cis436.update(
        { UserId: "1" },
        {
                UserId: "2",
                ProductId: 300,
                Ratings:4
        },
                {upsert: true }
)
```

(Here, the update command updates the collection with new data.)

MongoDB commands

3. Delete/Remove records from MongoDB collection

Code:

db.cis436.drop(UserId: '9')

(Here, the drop command drops data from the collection.)

4. Find/Search records from MongoDB Collection

Code:

db.cis436.find(UserId: '2')

(Here, the find command helps search the collection for the data one wants to search.)

Mahout algorithms & use cases

Algorithm #1: Decision Forest to classify data

```
$HADOOP_HOME/bin/hadoop jar $MAHOUT_HOME/examples/target/mahout-examples-<version>-job.jar
org.apache.mahout.classifier.df.mapreduce.TestForest -i nsl-kdd/KDDTest+.arff -ds nsl-kdd/KDDTrain+.info -m nsl-forest -a
-mr -o predictions
```

Use Case #1: Can be used to solve classification problems such as deciding whether to buy or not buy, whether to build or not etc.

Algorithm #2: Hidden Markov Models

```
$ echo "0 1 2 2 2 1 1 0 0 3 3 3 2 1 2 1 1 1 1 2 2 2 0 0 0 0 0 0 2 2 2 0 0 0 0 0 0 2 2 2 3 3 3 3 3 3 2 3 2 3 2 3 2 1 3 0 0 0 1 0 1 0 2 1
2 1 2 1 2 3 3 3 3 2 2 3 2 1 1 0" > hmm-input
$ export MAHOUT_LOCAL=true
$ $MAHOUT_HOME/bin/mahout baumwelch -i hmm-input -o hmm-model -nh 3 -no 4 -e .0001 -m 1000
$ $MAHOUT_HOME/bin/mahout hmmpredict -m hmm-model -o hmm-predictions -l 10
```

Use Case #2: Can be used for speech recognition, natural language processing etc.

Mahout algorithms & use cases

Algorithm #3: K-Means clustering

```
// run the CanopyDriver job
CanopyDriver.runJob("testdata", "output"
ManhattanDistanceMeasure.class.getName(), (float) 3.1, (float) 2.1, false);

// now run the KMeansDriver job
KMeansDriver.runJob("testdata", "output/clusters-0", "output",
EuclideanDistanceMeasure.class.getName(), "0.001", "10", true);

bin/mahout kmeans \
   -i <input vectors directory> \
   -c <input clusters directory> \
   -o <output working directory> \
   -k <optional number of initial clusters to sample from input vectors> \
   -dm <DistanceMeasure> \
   -x <maximum number of iterations> \
   -cd <optional convergence delta. Default is 0.5> \
   -ow <overwrite output directory if present>
   -cl <run input vector clustering after computing Canopies>
   -xm <execution method: sequential or mapreduce>
```

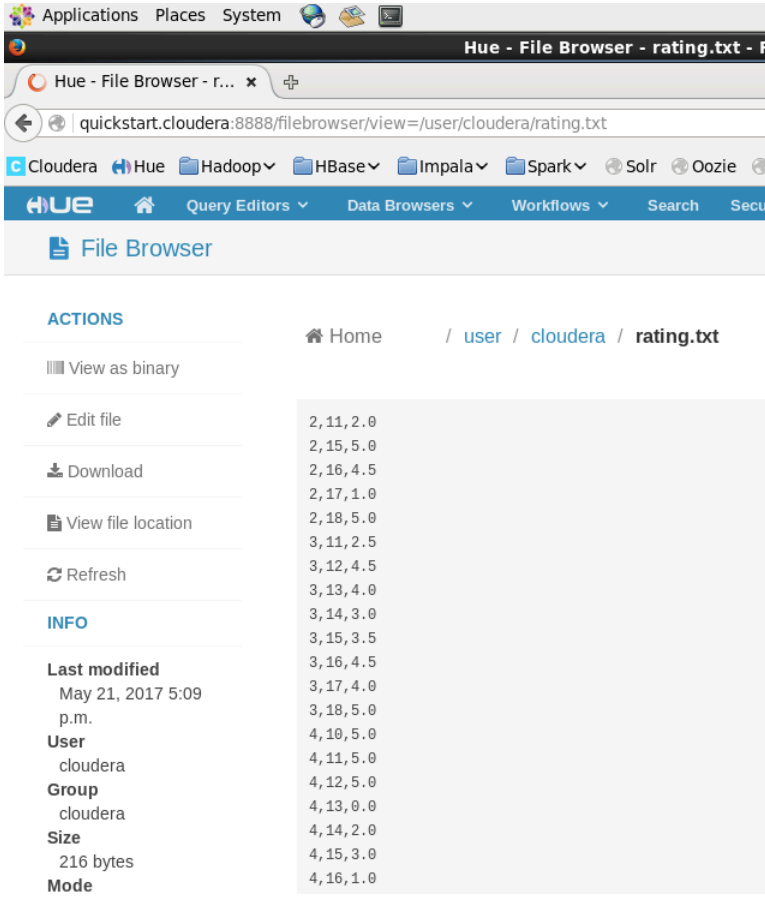
Use Case #3: Can be used to compute initial clusters for k-Kmeans in order to cluster data to find marketing segments, customer profile etc.

Build recommendation using Apache Mahout

Step 1:

Prepare and load data on Hadoop dataset in form user id, prod id, ratings:

-> The data was copied from the provided dataset and pasted in a file using the file browser interface.



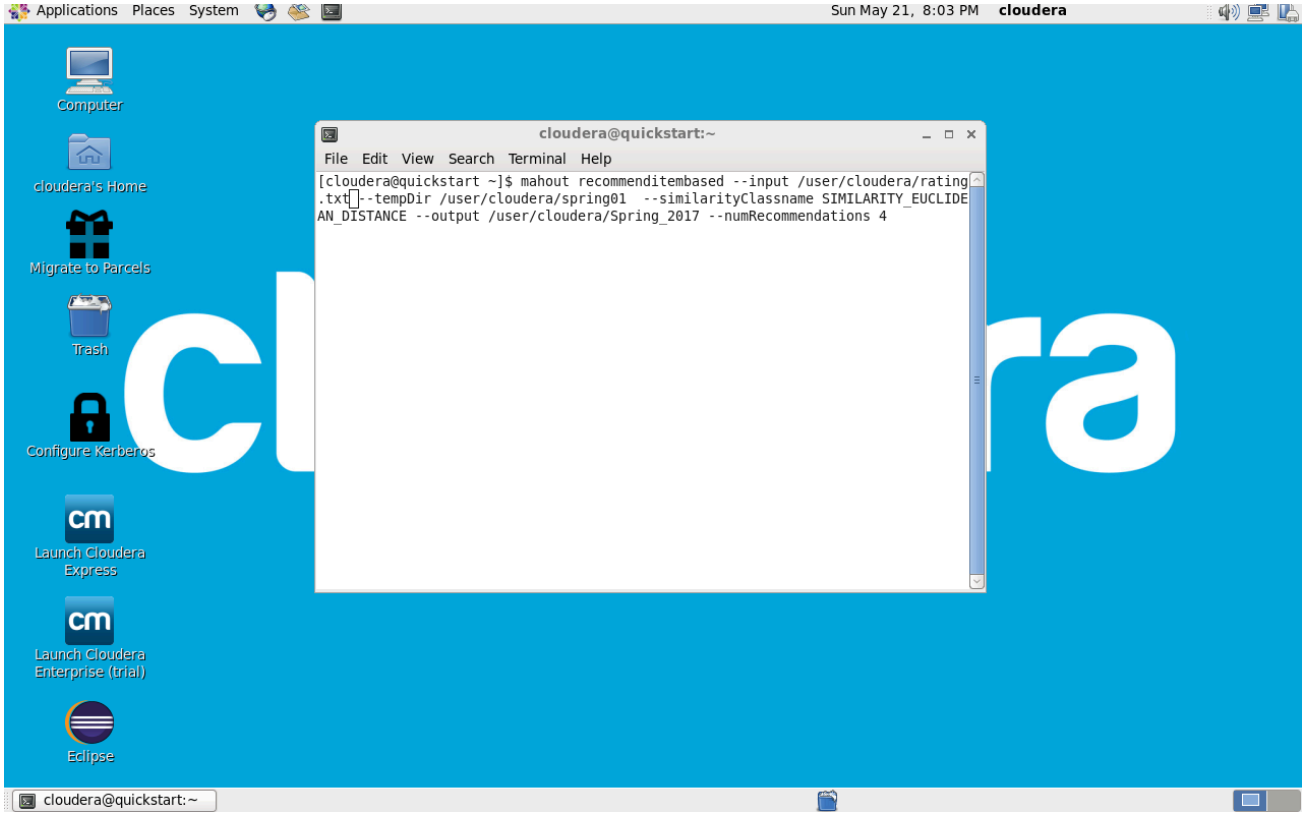
Build recommendation using Apache Mahout

Step 2:

Select and describe details of recommendation engine your are building

-> Below codes were used to create an item based recommendation engine in Apache Mahout. The item based recommendation engine was picked because of its versatility and ability to recommend the products based on the available data.

Codes used:
mahout recommenditembased --input /user/cloudera/recdemodata --tempDir /user/cloudera/sp1701 - similarityClassname SIMILARITY_EUCLIDEAN_DISTANCE --output /user/cloudera/sp_out_1701

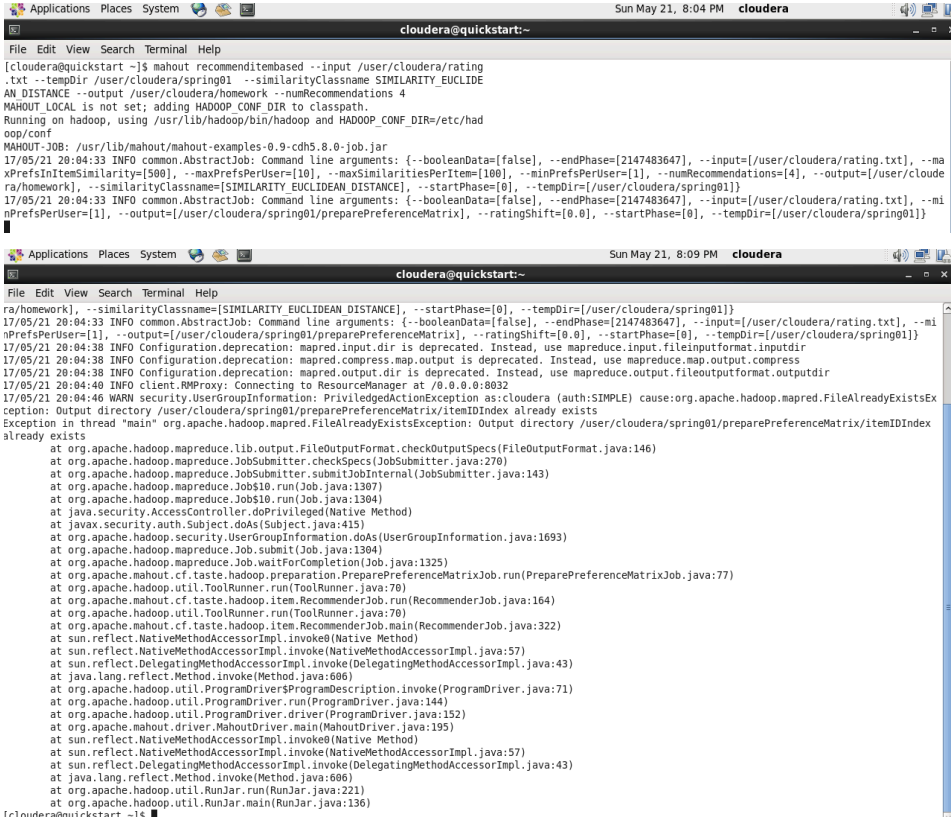


Build recommendation using Apache Mahout

Step 3:

Run recommendation engine using Mahout:

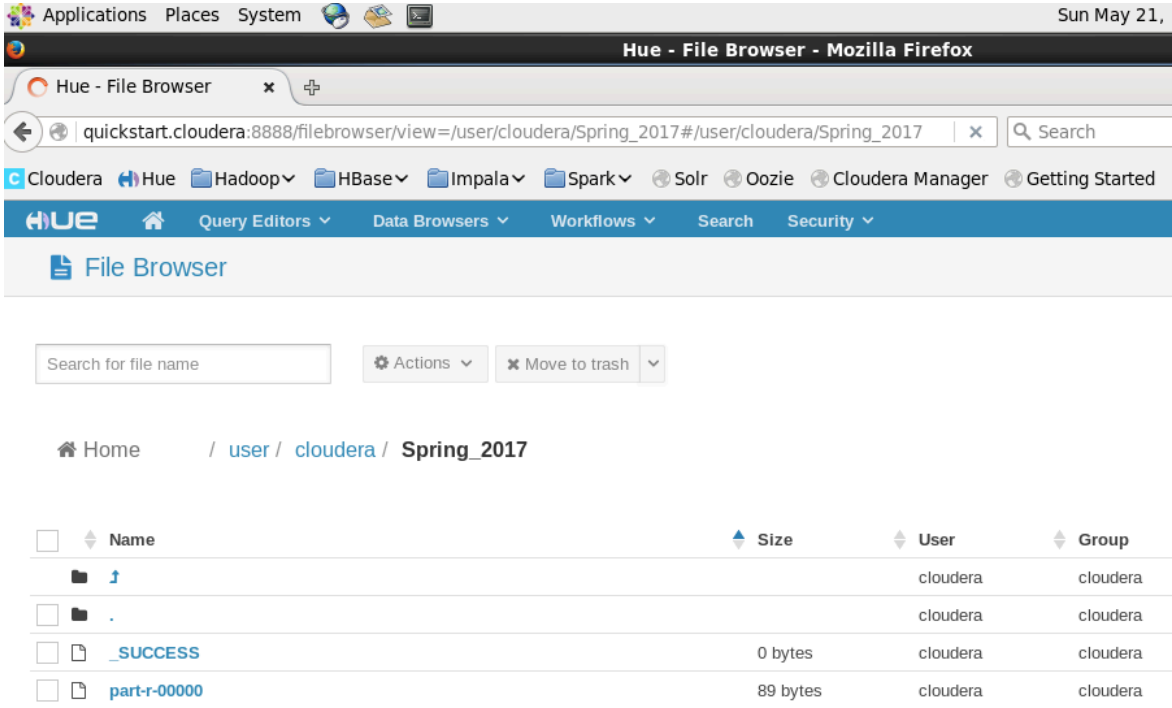
-> Upon running the recommendation engine with the codes in previous slide, the following run-message showed up on the screen. The output are discussed in next slides.



Build recommendation using Apache Mahout

Step 4:

Output validation: Upon finishing the run, we were able to cross validate and check that the run was successful. The screenshot on the right shows the success message and output files.

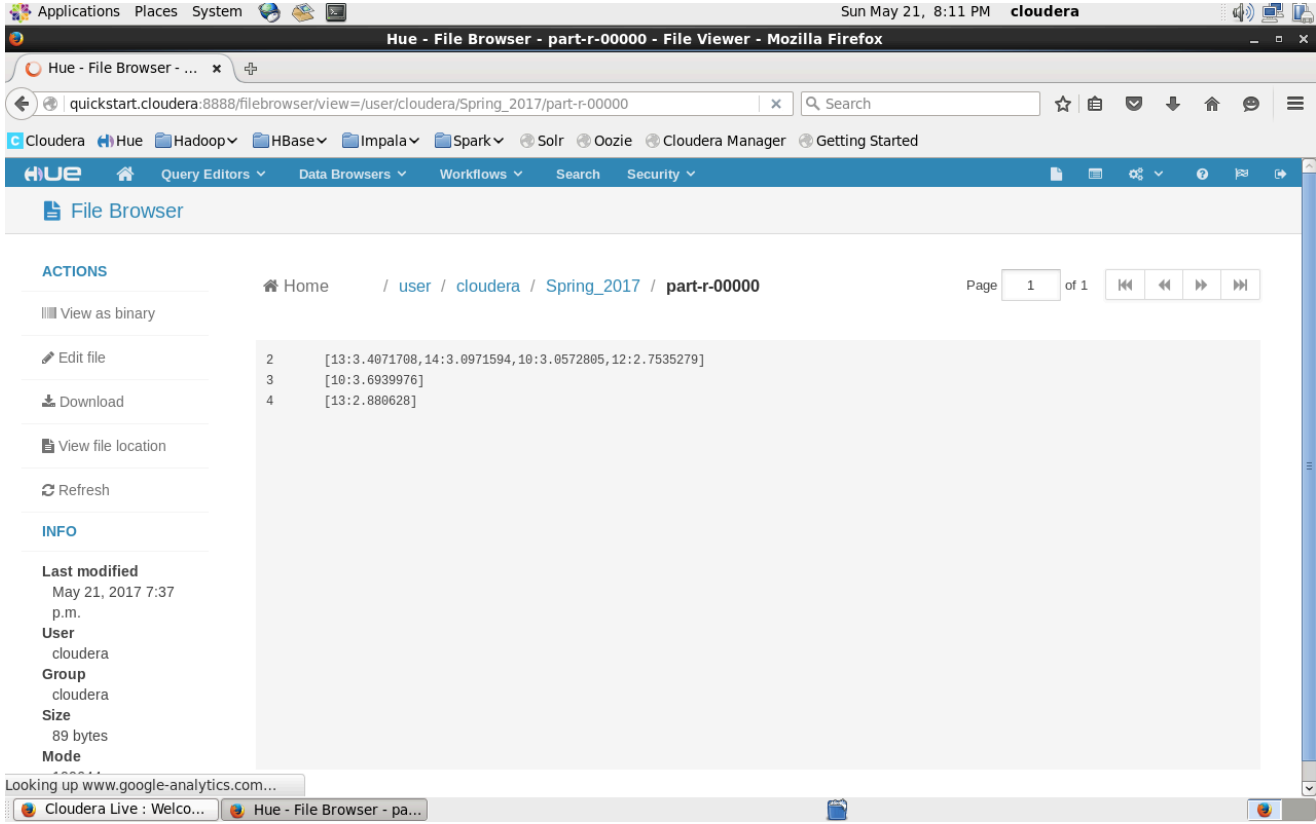


Build recommendation using Apache Mahout

Step 4:

Output explanation:

The output on the right shows the three recommended products. The recommended products are 10 and 13 with their respective ratings immediately separated by comma.

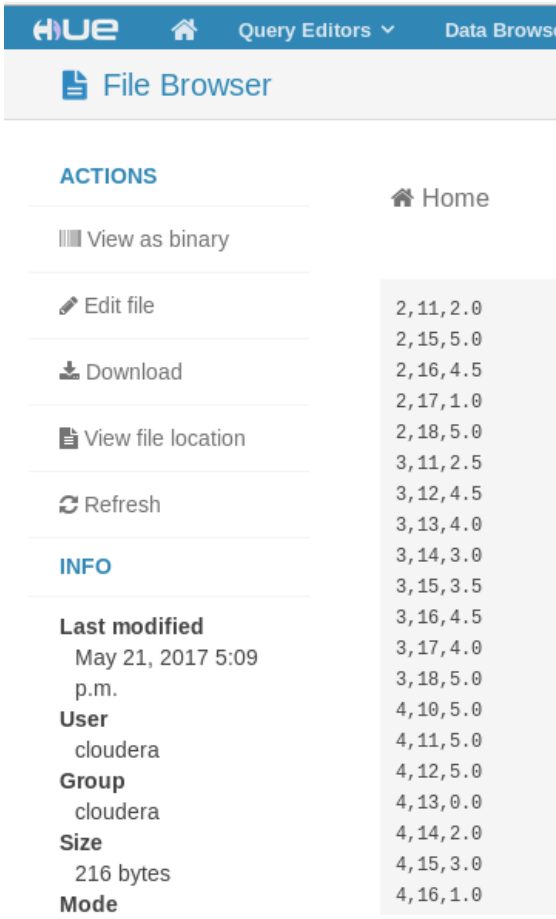


Build recommendation using Apache Mahout

Step 5:

Comparison:

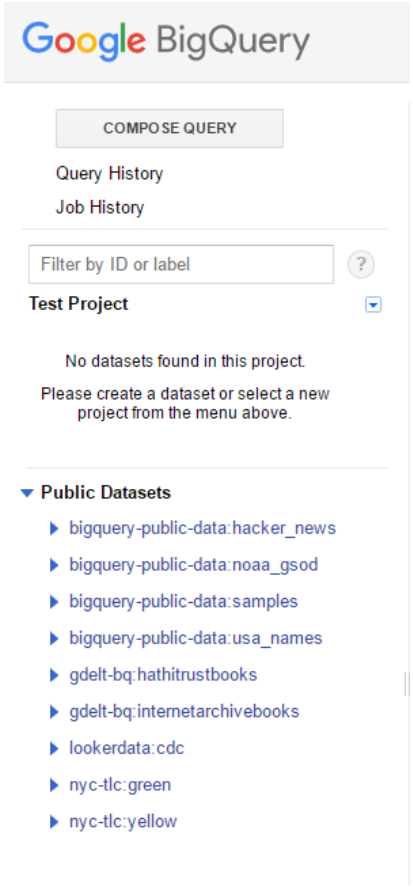
The screenshot on the right is the input file and the screenshot on the left is the output file. The input file has products from 11 thru 18 with different ratings. Based on the input file, we used Apache Mahout's item based recommendation engine to recommend three additional products with different ratings. The results on the left are the new products with their respective ratings that are recommended based on the data to the right.



Analytics on Cloud using Google Big Query

Create Google Big Query account and show list of public tables, which table you liked most

-> We created a Google Big Query account and found the list of public tables screenshot on the right. I found the US census data most helpful amongst all of the data sets available.



Analytics on Cloud using Google Big Query

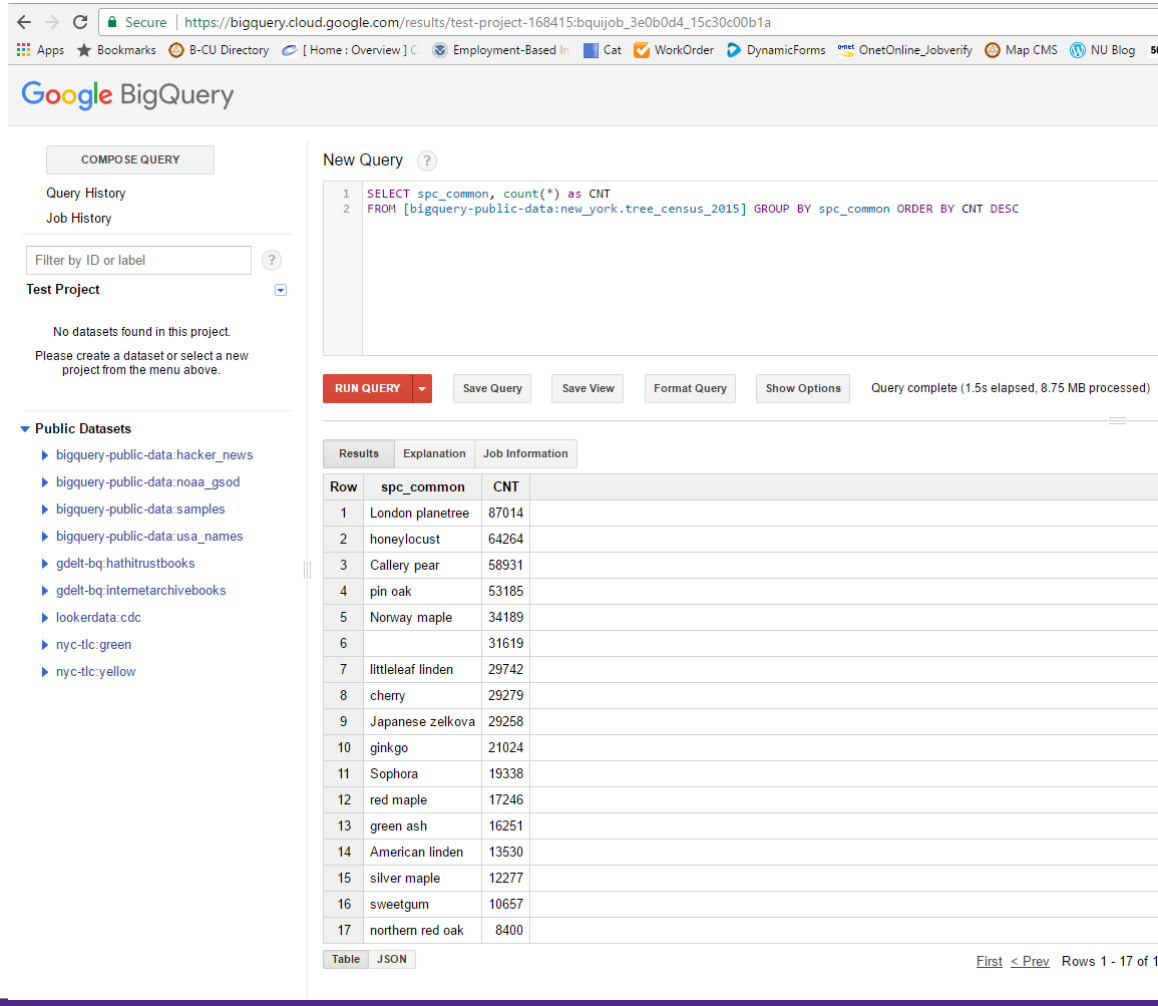
Run query and share results, explain what this query does?

Codes used:

SELECT spc_common, count(*) as CNT
FROM [bigquery-public-data:new_york.tree_census_2015] GROUP BY spc_common ORDER BY CNT DESC

Result explanation:

The query resulted in the count of the number of line items in the new.york.tree.census_2015 dataset grouped by the type of spc_common. We then ordered the data in descending order based on the count of spc_common. It appears that London plantree, honeylocust and callery,pear were most common in 2015.



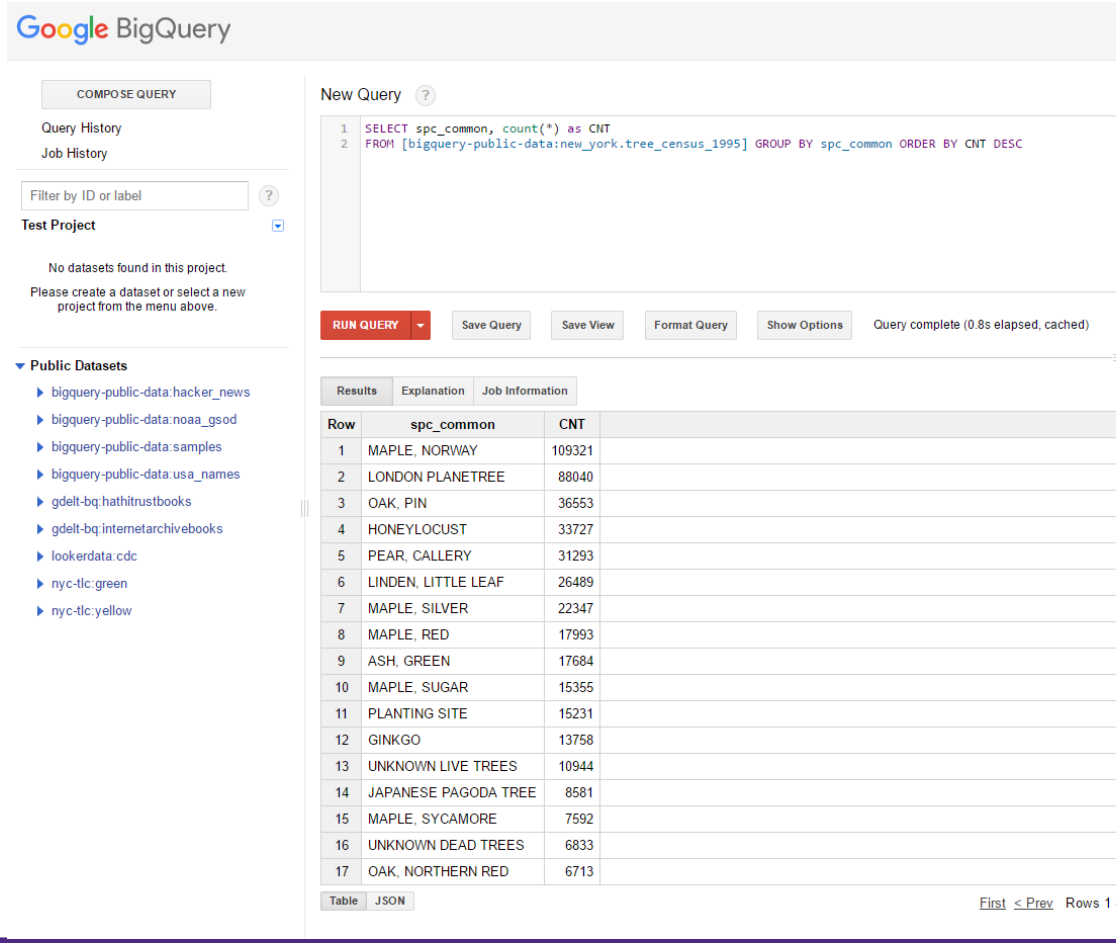
Analytics on Cloud using Google Big Query

Change query to run for 1995 Census and explain what changes you see since 1995 to 2015 in New york city.

Codes Used:
SELECT spc_common, count(*) as CNT
FROM [bigquery-public-data:new_york.tree_census_1995] GROUP BY spc_common ORDER BY CNT DESC

Results:

When changing the year in a similar data set from previous slide, the result was entirely different. Unlike 2015, it appears that Maple_Norway, London_Planetree and Oak,Pin had the highest census in 1995.



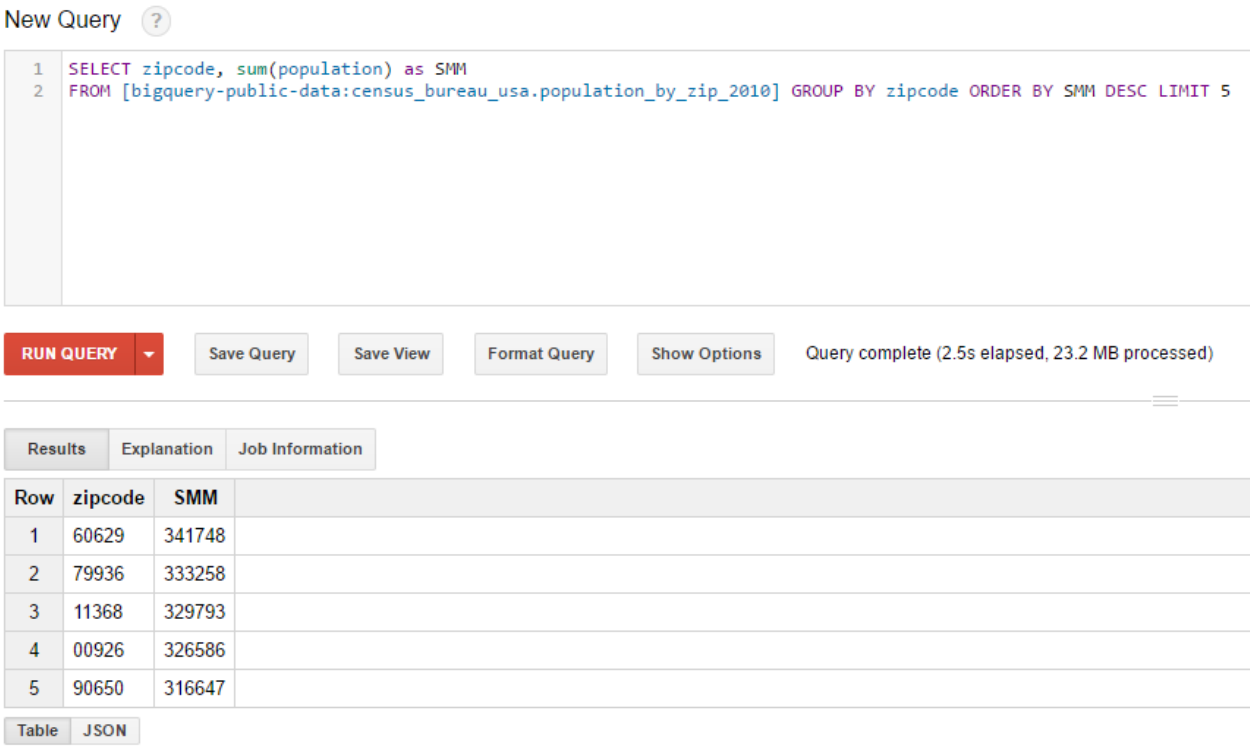
Analytics on Cloud using Google Big Query

BONUS Question #1

2. List top 5 zipcodes with highest population in USA USE table census bureau_usa in 2010

Codes Used:
SELECT zipcodes, sum(population) as SMM FROM [bigquery-public-data:census_bureau_usa.population_by_zip_2010] GROUP BY zipcode ORDER BY SMM DESC LIMIT 5

Results are presented on the screenshot to the right.



Analytics on Cloud using Google Big Query

BONUS Question #2

1. List top 5 zipcodes with highest population in USA USE table census_bureau_usa in 2000

Codes Used:
SELECT zipcodes, sum(population) as SMM FROM [bigquery-public-data:census_bureau_usa.population_by_zip_2000] GROUP BY zipcode ORDER BY SMM DESC LIMIT 5

Results are presented on the screenshot to the right.

New Query

```
1  SELECT zipcode, sum(population) as SMM
2  FROM [bigquery-public-data:census_bureau_usa.population_by_zip_2000] GROUP BY zipcode ORDER BY SMM DESC LIMIT 5
```

RUN QUERY | Save Query | Save View | Format Query | Show Options | Query complete (2.6s elapsed, 22.7 MB processed)

Results | Explanation | Job Information

Row	zipcode	SMM
1	00725	431961
2	78572	342372
3	60629	341952
4	60623	324432
5	11226	318462

Table | JSON

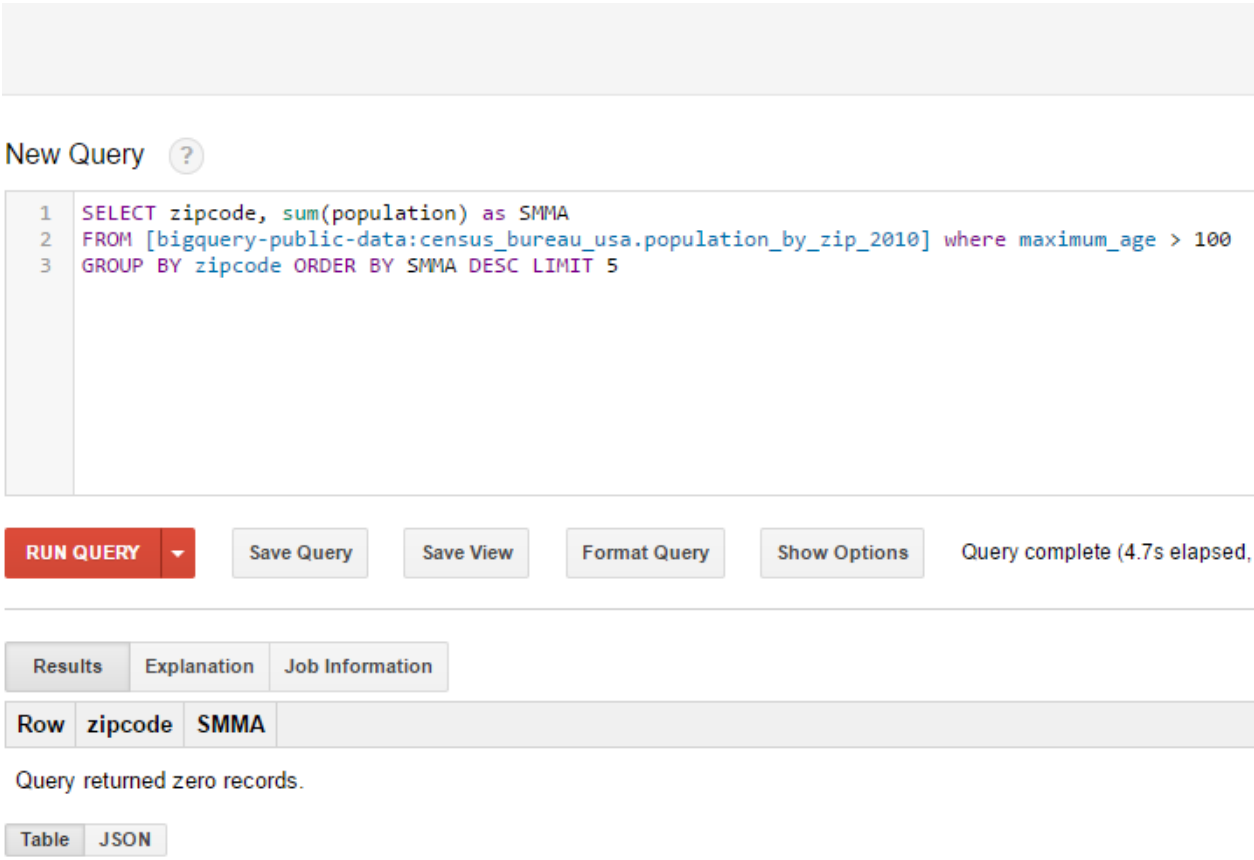
Analytics on Cloud using Google Big Query

BONUS Question #3

3. Per 2010 census which zipcode has more number of peoples with max age >100

Codes Used:
SELECT zipcodes, sum(population) as SMMA FROM [bigquery-public-data:census_bureau_usa.population_by_zip_2010] where maximum_age > 100 GROUP BY zipcode ORDER BY SMMA DESC LIMIT 5

Apparently, there were no entries in 2010 census data that has a zip code with number of peoples whose max age is greater than 100.



Thank You!