ZEPPELIN INSTALLATION

docker run -p 8080:8080 --rm -v /Users/rajdaiya/Documents/PBDA\:/shared/users/raj --name zeppelin apache/zeppelin:0.7.3

PIG QUERIES (Pig Output in solution/ans.tsv)

```
register '/shared/users/raj/piggybank-0.11.0.jar';
register '/shared/users/raj/elephant-bird-core-4.15.jar';
register '/shared/users/raj/elephant-bird-hadoop-compat-4.15.jar';
register '/shared/users/raj/elephant-bird-pig-4.15.jar';
register '/shared/users/raj/json-simple-1.1.1.jar';
```

Query 1: Summarize the number of unique *reviewers* by US city, by business category. That is, count the unique reviewers by city, by business.

```
businesses = LOAD '/shared/users/raj/dataset/business.json' using
com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') as json:map[];

tip = LOAD '/shared/users/raj/dataset/tip.json' using
com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') as json:map[];

uscities = LOAD '/shared/users/raj/dataset/uscities.csv' USING PigStorage(',') AS (a:chararray,
b:chararray, c:chararray, d:chararray, e:chararray, f:chararray,
g:chararray, h:chararray, i:chararray, j:chararray, k:chararray, l:chararray);

guscities = FOREACH(GROUP uscities BY (c)) GENERATE FLATTEN(group) AS st;

businesses = FOREACH businesses GENERATE json#'business_id' as business_id,json#'name'
as name,json#'neighborhood' as neighborhood,json#'address' as address,json#'city' as
city,json#'state' as state,json#'postal_code' as postal_code,json#'latitude' as
latitude,json#'longitude' as longitude,json#'stars' as stars,json#'review_count' as
review_count,json#'is_open' as is_open,json#'attributes' as attributes,json#'categories' as
categories,json#'hours' as hours,json#'type' as type;

businesses_uscities = JOIN guscities by st LEFT OUTER, businesses BY state;

attributes = FOREACH businesses_uscities GENERATE (int)review_count AS review_count,
city AS city, FLATTEN(categories) as categories;

groupcitycat = GROUP attributes BY (city, categories);

reviewcounts = FOREACH groupcitycat GENERATE FLATTEN(group) as (city, categories),
COUNT(attributes.review_count) as total_count;

orderreviewcount = ORDER reviewcounts BY city;
```

```
STORE orderreviewcount INTO '/shared/users/raj/solution/ans1.tsv';
```

Query 2: Rank all *cities* by # of stars descending, for each category

```
businesses = LOAD '/shared/users/raj/dataset/business.json' using
com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') as json:map[];

attributes = FOREACH businesses GENERATE (float)json#'stars' AS stars, json#'city' AS city,
FLATTEN(json#'categories') as categories;

groupcitycat = GROUP attributes BY (city,categories);

avgstars = FOREACH groupcitycat GENERATE AVG(attributes.stars) as st, FLATTEN(group)
as (city,categories);

result = RANK avgstars by categories ASC, st DESC;

STORE result INTO '/shared/users/raj/solution/ans2.tsv';
```

Query 3: What is the average rank (# stars) for businesses within 15 km of Edinburgh Castle, Scotland, by type of business (category)?

```
businesses = LOAD '/shared/users/raj/dataset/business.json' using
com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') as json:map[];


business_distance= FOREACH businesses GENERATE FLATTEN(json#'categories') as
categories,(double)json#'stars' as stars, json#'business_id' AS business_id, json#'name' AS name,
json#'city' AS city, json#'latitude' AS latitude, json#'longitude' AS
longitude,ACOS(SIN(55.9469753*3.14159/180)*SIN((json#'latitude')*3.14159/180)+COS(55
.9469753*3.14159/180)*COS((json#'latitude')*3.14159/180)*COS(-3.2096308*3.14159/180 -
(json#'longitude')*3.14159/180))*6371 as distance;

business_distance= FILTER business_distance BY distance<15;


filteredDataForQ5 = business_distance;

groupedData = GROUP business_distance BY categories;

finalData = FOREACH groupedData GENERATE group as
category,AVG(business_distance.stars);

STORE finalData INTO '/shared/users/raj/solution/ans3.tsv';
```

Query 4: Rank reviewers in Q3 by their number of reviews. For the top 10 reviewers, show their average number of stars, by category.

```
businesses = LOAD '/shared/users/raj/dataset/business.json' using
com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') as json:map[];

business_distance= FOREACH businesses GENERATE FLATTEN(json#'categories') as
categories,(double)json#'stars' as stars, json#'business_id' AS business_id, json#'name' AS name,
json#'city' AS city, json#'latitude' AS latitude, json#'longitude' AS
longitude,ACOS(SIN(55.9469753*3.14159/180)*SIN((json#'latitude')*3.14159/180)+COS(55
.9469753*3.14159/180)*COS((json#'latitude')*3.14159/180)*COS(-3.2096308*3.14159/180 -
(json#'longitude')*3.14159/180))*6371 as distance;

business_distance= FILTER business_distance BY distance<=15;
filteredDataForQ5 = business_distance;

data_user = LOAD '/shared/users/raj/dataset/user.json' USING
com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') AS (json:map[]);

data_review = LOAD '/shared/users/raj/dataset/review.json' USING
com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') AS (json:map[]);

at_user = FOREACH data_user GENERATE json#'user_id' AS user_id, (int)json#'review_count'
as review_count;

at_review = FOREACH data_review GENERATE json#'user_id' AS user_id, json#'review_id' as
review_id, json#'business_id' as business_id,(float)json#'stars' AS stars;

rank_users = ORDER at_user BY review_count DESC;

rank_limit = LIMIT rank_users 10;

rev_bus = JOIN at_review BY business_id, filteredDataForQ5 BY business_id;

total_combine = JOIN rank_limit BY user_id, rev_bus BY at_review::user_id;

get_reqd = FOREACH total_combine GENERATE at_review::user_id,
filteredDataForQ5::categories, at_review::star;

grouping = GROUP get_reqd by (at_review::user_id, filteredDataForQ5::categories);

avg_stars = FOREACH grouping GENERATE FLATTEN(group) as (usr, cat),
AVG(get_reqd.star) as star;

STORE avg_stars INTO '/shared/users/raj/solution/ans4.tsv';
```

Query 5: For the top 10 and bottom 10 category *Food* businesses in Q3, (in terms of stars), summarize star rating for reviews in January through May only.

```
businesses = LOAD '/shared/users/raj/dataset/business.json' using
com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') as json:map[];

business_distance= FOREACH businesses GENERATE FLATTEN(json#'categories') as
categories,(double)json#'stars' as stars, json#'business_id' AS business_id, json#'name' AS name,
json#'city' AS city, json#'latitude' AS latitude, json#'longitude' AS
longitude,ACOS(SIN(55.9469753*3.14159/180)*SIN((json#'latitude')*3.14159/180)+COS(55.9
469753*3.14159/180)*COS((json#'latitude')*3.14159/180)*COS(-3.2096308*3.14159/180 -
(json#'longitude')*3.14159/180))*6371 as distance;
business_distance= FILTER business_distance BY distance<=15;
filteredDataForQ5 = business_distance;


reviewdata = LOAD '/shared/users/raj/dataset/review.json' USING
com.twitter.elephantbird.pig.load.JsonLoader('-nestedLoad') AS (json:map[]);

filterlatlong = filter filteredDataForQ5 by categories == 'Food';

topfilter = ORDER filterlatlong BY stars DESC;

bottomfilter = ORDER filterlatlong BY stars ASC;

topele = LIMIT topfilter 10;

bottomele = LIMIT bottomfilter 10;

topbottomunion = UNION topele, bottomele;

rev_attributes = FOREACH reviewdata GENERATE json#'business_id' AS business_id,
(datetime)json#'date' AS date, json#'review_id' AS review_id, (float)json#'stars' AS stars,
json#'user_id' AS user_id;

monthfilter = FILTER rev_attributes BY (GetMonth(date)==1) OR (GetMonth(date)==2) OR
                (GetMonth(date)==3) OR (GetMonth(date)==4) OR (GetMonth(date)==5);

combined = JOIN topbottomunion BY business_id, monthfilter BY business_id;

comb_result = FOREACH combined GENERATE monthfilter::stars, monthfilter::business_id,
                monthfilter::review_id;

res = GROUP comb_result by monthfilter::business_id;
```

final = FOREACH res GENERATE group as bus_id, AVG(comb_result.stars) as star;

STORE final INTO '/shared/users/raj/solution/ans5.tsv';

SCALA QUERIES IN APACHE SPARK

1. import scala.collection.mutable.WrappedArray

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

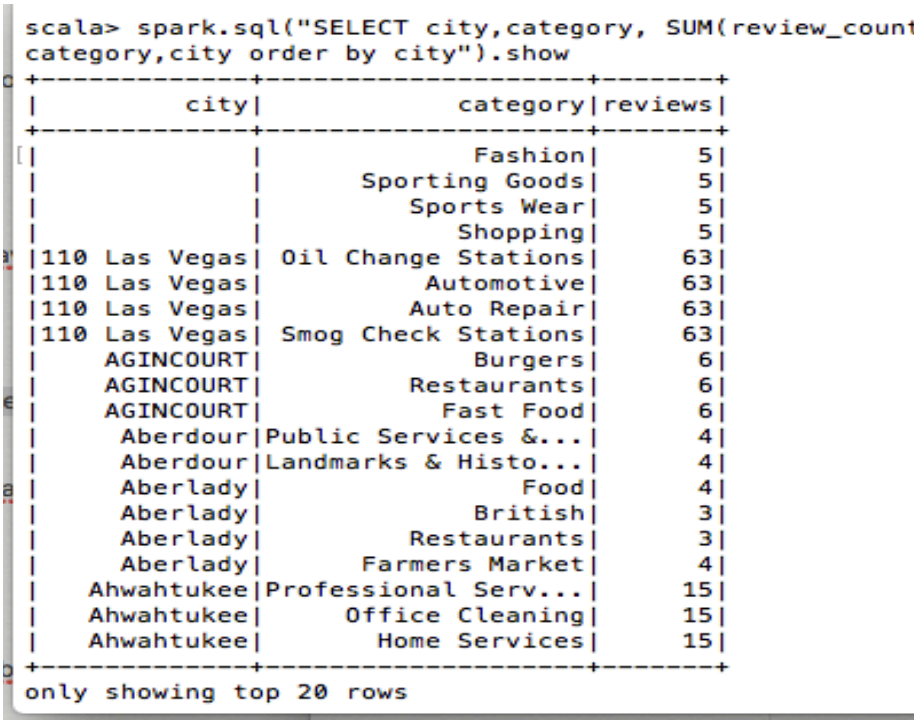
   val business = spark.read.json("/Users/Raj/Downloads/dataset/business.json")

   val ans1 = business.withColumn("category", explode(

   when(col("categories").isNotNull,col("categories") .otherwise(array(lit(null).cast("string")
))))

   ans1.registerTempTable("business")

   spark.sql("SELECT city,category, SUM(review_count) AS total_review FROM business group by category,city order by city").show



```
scala> spark.sql("SELECT city,category, SUM(review_count
category,city order by city").show
+-------------+--------------------+-------+
|         city|            category|reviews|
+-------------+--------------------+-------+
|             |             Fashion|      5|
|             |      Sporting Goods|      5|
|             |         Sports Wear|      5|
|             |            Shopping|      5|
|110 Las Vegas| Oil Change Stations|     63|
|110 Las Vegas|          Automotive|     63|
|110 Las Vegas|         Auto Repair|     63|
|110 Las Vegas| Smog Check Stations|     63|
|    AGINCOURT|             Burgers|      6|
|    AGINCOURT|         Restaurants|      6|
|    AGINCOURT|           Fast Food|      6|
|     Aberdour|Public Services &...|      4|
|     Aberdour|Landmarks & Histo...|      4|
|     Aberlady|                Food|      4|
|     Aberlady|             British|      3|
|     Aberlady|         Restaurants|      3|
|     Aberlady|      Farmers Market|      4|
|   Ahwahtukee|Professional Serv...|     15|
|   Ahwahtukee|     Office Cleaning|     15|
|   Ahwahtukee|       Home Services|     15|
+-------------+--------------------+-------+
only showing top 20 rows
```

```
import scala.collection.mutable.WrappedArray
import spark.implicits._
import org.apache.spark.sql.functions._
val business = spark.read.json("/Users/Raj/Downloads/dataset/business.json")

 val b = business.withColumn("category", explode(

when(col("categories").isNotNull,col("categories")).otherwise(array(lit(null).cast("string"
)))))

b.registerTempTable("business")
val df = sqlContext.sql("SELECT  category,city,avg(stars) as avg_stars from business
group by category,city order by category asc, avg_stars desc")

df.write.csv("/Users/Raj/Downloads/dataset/ans2.csv")
```

```
+----------------+-------------+------------------+
|        category|         city|         avg_stars|
+----------------+-------------+------------------+
|       & Probates|    Scottsdale|               5.0|
|       & Probates|      Glendale|               5.0|
|       & Probates|     Champaign|               5.0|
|       & Probates|New Kensington|               5.0|
|       & Probates|       Peoria|               5.0|
|       & Probates|         Mesa|              4.75|
|       & Probates|       Gilbert|              4.75|
|       & Probates|        Tempe|               4.5|
|       & Probates|       Phoenix|               4.5|
|       & Probates|     Las Vegas| 4.291666666666667|
|       & Probates|     Henderson|             4.125|
|       & Probates|    Pittsburgh|               4.0|
|       & Probates|      Chandler|               4.0|
|      3D Printing|     Henderson|               5.0|
|      3D Printing|       Gilbert|               3.5|
|      3D Printing|       Toronto|3.2857142857142856|
|ATV Rentals/Tours|    Scottsdale|               5.0|
|ATV Rentals/Tours|       Phoenix|               5.0|
|ATV Rentals/Tours|      Sun City|               5.0|
|ATV Rentals/Tours|      Carnegie|               5.0|
+----------------+-------------+------------------+
only showing top 20 rows
```

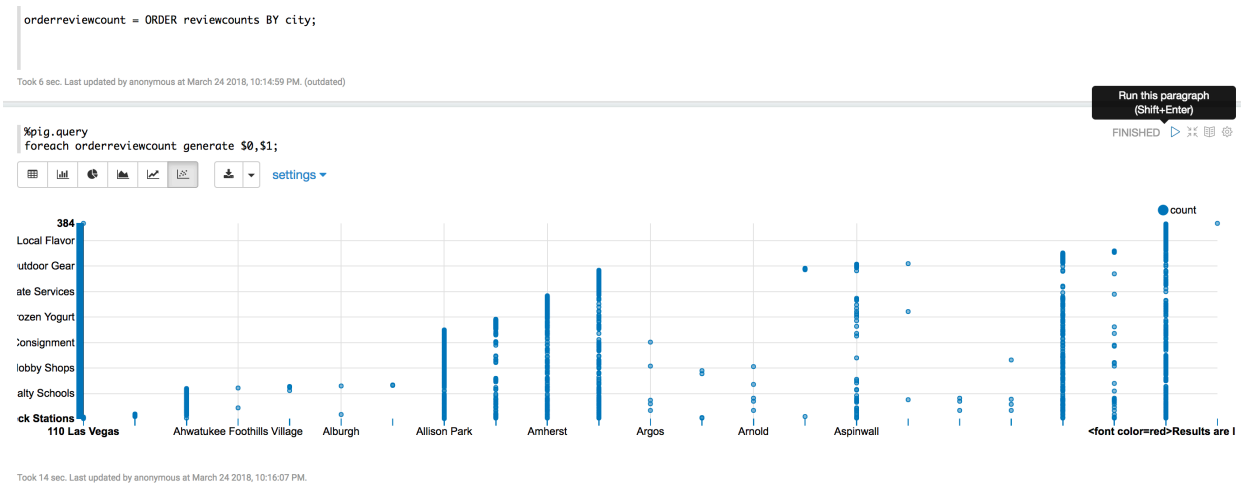
```
import scala.collection.mutable.WrappedArray
 import spark.implicits._
import org.apache.spark.sql.functions._

val business = spark.read.json("/Users/Raj/Downloads/dataset/business.json")
val lat_long_business = spark.sql("select *, (latitude*3.14/180) lat_rad,
(longitude*3.14/180) long_rad from business")

lat_long_business.createOrReplaceTempView("lat_long_business")

spark.sql("SELECT categories, avg(stars), avg(review_count) FROM lat_long_business
WHERE ACOS( SIN(lat_rad) * SIN(0.76189206903) + COS(lat_rad) *
```

COS(0.76189206903) * COS(long_rad + 1.385498211)) * 6371 <= 15 group by categories").show

```
Kinjal — java ◂ spark-shell — 93×28
scala> spark.sql("SELECT categories, avg(stars) FROM BUSINESS_EXPLODE_LAT_LONG_RAD WHERE ACOS
( SIN(lat_rad) * SIN(0.76189206903) + COS(lat_rad) * COS(0.76189206903) * COS(long_rad + 1.38
5498211 )) * 6371 <= 15 group by categories").show
+--------------------+------------------+
|          categories|        avg(stars)|
+--------------------+------------------+
|[Health & Medical...|               4.0|
|[Dog Walkers, Pet...|               5.0|
|[Restaurants, Car...|               4.0|
|[Vietnamese, Fast...|               3.5|
|[Coffee & Tea, Co...|               4.0|
|[Eyewear & Optici...|               4.0|
|[Oil Change Stati...|               1.0|
|[Shopping, Linger...|               4.5|
|[Beauty & Spas, H...|               2.0|
|[Health & Medical...|               5.0|
|[Fashion, Shoppin...|              3.75|
|[Specialty Food, ...|               4.5|
|[Printing Service...|               5.0|
|[Arts & Entertain...|               3.0|
|[Books, Mags, Mus...|               3.5|
|[Food, Food Deliv...|2.9166666666666665|
|[Threading Servic...|               3.0|
|[Chinese, Restaur...|               3.5|
|[Hair Removal, Ha...|               3.0|
|[Auto Parts & Sup...|               5.0|
+--------------------+------------------+
only showing top 20 rows
```

VISUALIZATION of Pig Scripts using Zeppelin

Query 1:



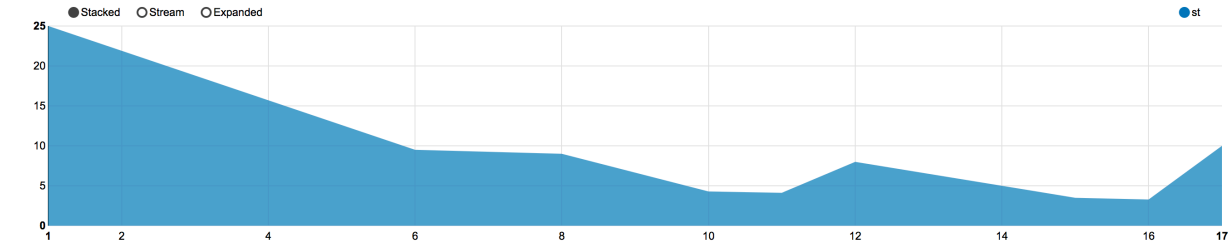
Query 2:

ok 3 sec. Last updated by anonymous at March 24 2018, 10:08:34 PM.

```
%pig.query
foreach vis generate $0,$1;
```

FINISHED

settings



ok 11 sec. Last updated by anonymous at March 24 2018, 10:08:47 PM. (outdated)

Query 3:

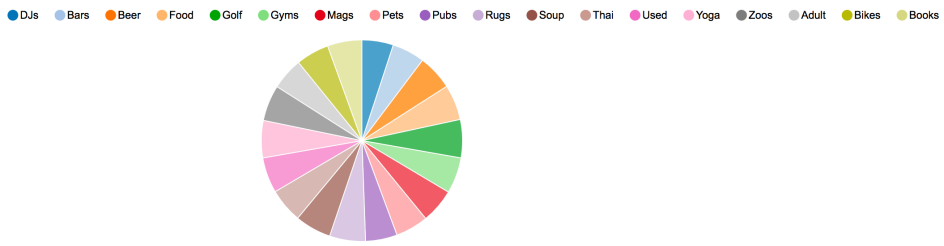
Took 3 sec. Last updated by anonymous at March 24 2018, 10:09:38 PM.

```
%pig.query
foreach vis generate $0,$1;
```

FINISHED

settings

Pie Chart



Query 4:

```
avg_stars = FOREACH grouping GENERATE FLATTEN(group) as (usr, cato), AVG(get_reqd.star) as star;
```

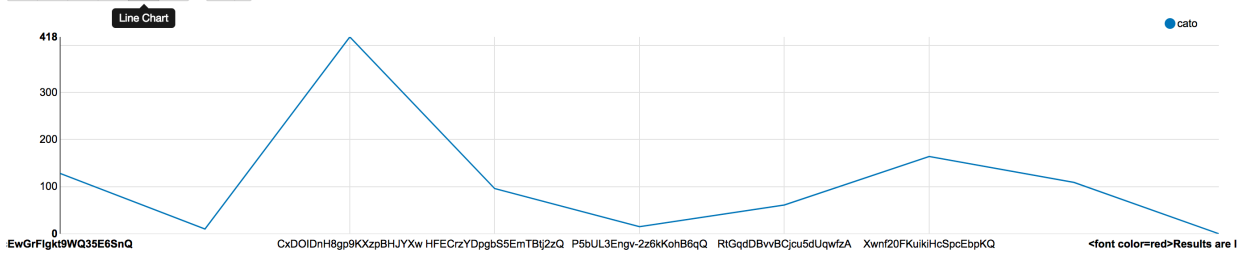
Took 7 sec. Last updated by anonymous at March 24 2018, 10:18:26 PM.

```
%pig.query
foreach avg_stars generate $0,$1;
```

FINISHED

settings

Line Chart



Took 3 min 43 sec. Last updated by anonymous at March 24 2018, 10:22:16 PM. (outdated)

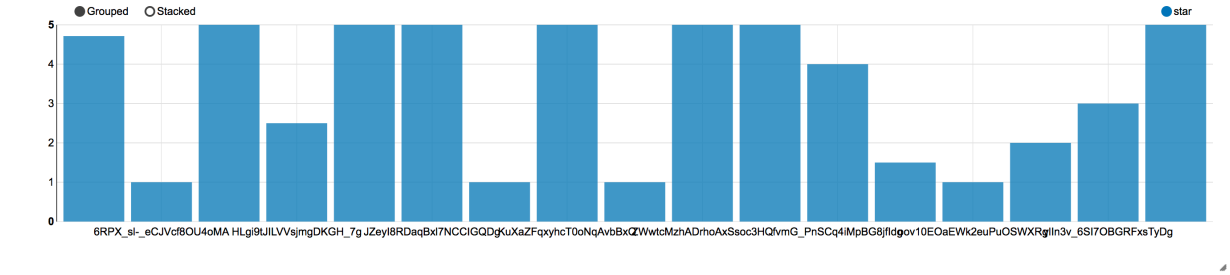
Query 5:

```
vis= LIMIT final 18;
```

Took 8 sec. Last updated by anonymous at March 24 2018, 10:10:54 PM.

```
%pig.query
foreach vis generate $0,$1;
```

FINISHED



Took 2 min 1 sec. Last updated by anonymous at March 24 2018, 10:12:58 PM. (outdated)