**Day 3**

|  |
| --- |
| ##Spark SQL test: review2 should be in hdfs for SparkSQL %pyspark review2 = sqlContext.sql("select \* from review2"); review2.show()  %pyspark review2.select("\*").show(10)  ## can find out the table schema from HDFS hive table %pyspark review2.printSchema()  ## Read json file and infer schema as well %pyspark business = sqlContext.read.option("inferSchema", "true").json("s3://hipicdatasets/yelp/business/business.json") business.select("\*").show(10)  ## can find out the table schema inferred from Json file => easy to build Hive table as reference %pyspark business.printSchema() |

|  |
| --- |
| // read all the json files **and** create dataframes val bizDF = spark.read.json("s3://hp-skt-emrbucket/yelp/business.json") val checkinDF = spark.read.json("s3://hp-skt-emrbucket/yelp/checkin.json") val photosDF = spark.read.json("s3://hp-skt-emrbucket/yelp/photos.json") val reviewDF = spark.read.json("s3://hp-skt-emrbucket/yelp/review.json") val tipDF = spark.read.json("s3://hp-skt-emrbucket/yelp/tip.json") val userDF = spark.read.json("s3://hp-skt-emrbucket/yelp/user.json") |

|  |
| --- |
| // try printing out the schema. Notice how easily we can get the schema. reviewDF.printSchema reviewDF.show // try saving a dataframe directly as a hive table reviewDF.write.mode("append").  option("path","s3://hp-skt-emrbucket/yelp/reviewDF2/").  saveAsTable("reviewDF2")  // try again with a complicated json file such as business.json  // were you able to create a hive table directly? |

|  |
| --- |
| // **create** **temporary** views so we can **use** them later **in** **SQL** statements bizDF.createTempView("business4\_df") checkinDF.createTempView("checkin\_df") photosDF.createTempView("photos\_df") reviewDF.createTempView("review\_df") tipDF.createTempView("tips\_df") userDF.createTempView("user\_df") |

Now, on a new line type the following to enter into SQL mode and get a list of all the tables.

|  |
| --- |
| %sql **show** **tables** |

Create the exploded dataframe. Three different styles are shown.

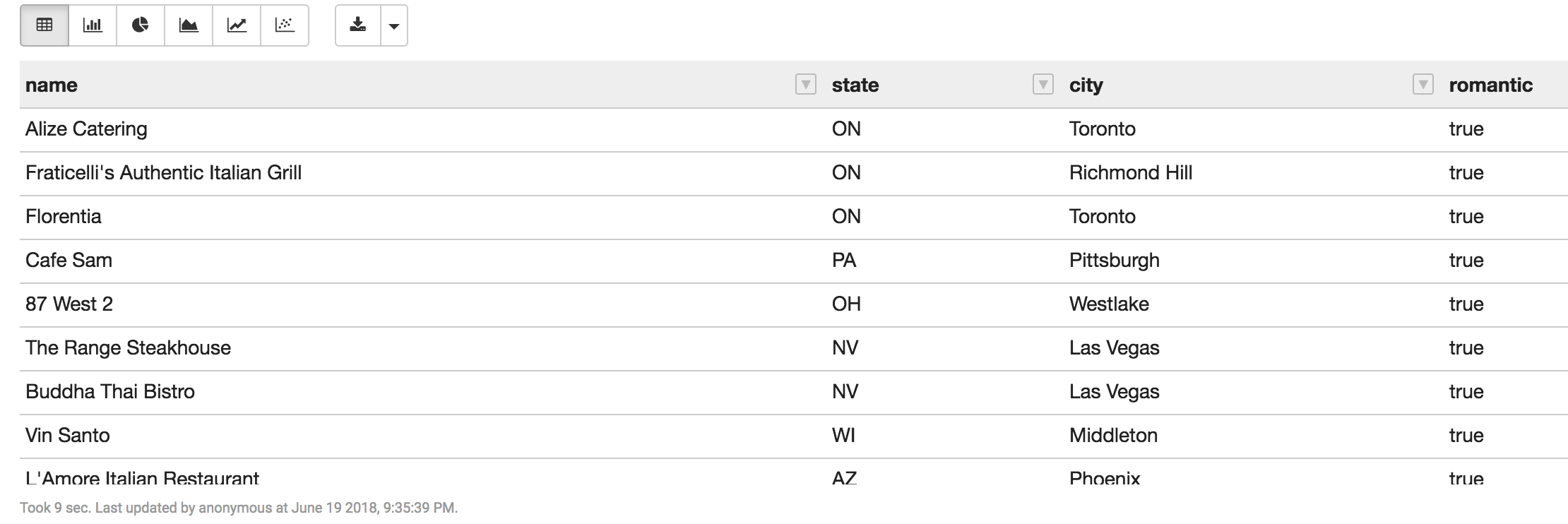
|  |
| --- |
| //**create** exploded **view** **of** business. **Use** **SQL** **as** well **as** **column** expression val explodedDF = spark.sql("""   SELECT \* FROM business4   LATERAL VIEW explode(categories) c AS cat\_exploded """) explodedDF.show() explodedDF.createTempView("exploded\_df") val exploded2DF = bizDF.withColumn("cat\_exploded", explode($"categories")) exploded2DF.show() val exploded3DF = bizDF.  selectExpr("\*","explode(categories) as cat\_explode") //**check** **to** make sure the three commands produced same **result** explodedDF.count exploded2DF.count exploded3DF.count |

Create the restaurants dataframe.

|  |
| --- |
| // create restaurant DF val restaurantsDF = spark.sql("""   SELECT \* FROM exploded\_df WHERE cat\_exploded="Restaurants"   """) restaurantsDF.createTempView("restaurants\_df") restaurantsDF.show() |

Let’s test querying a nested field.

|  |
| --- |
| %sql **SELECT** **name**, state, city, attributes.ambience.romantic romantic  **FROM** restaurants\_df  **WHERE** attributes.ambience.romantic = true **LIMIT** 10 |



Try again using column expressions.

|  |
| --- |
| val romanticDF = restaurantsDF.  select($"name",$"state",$"city",  ($"attributes.ambience.romantic").alias("romantic")).  where($"attributes.ambience.romantic" === true) romanticDF.show |

Now create the review of restaurants dataframe. Both SQL and column expression are shown.

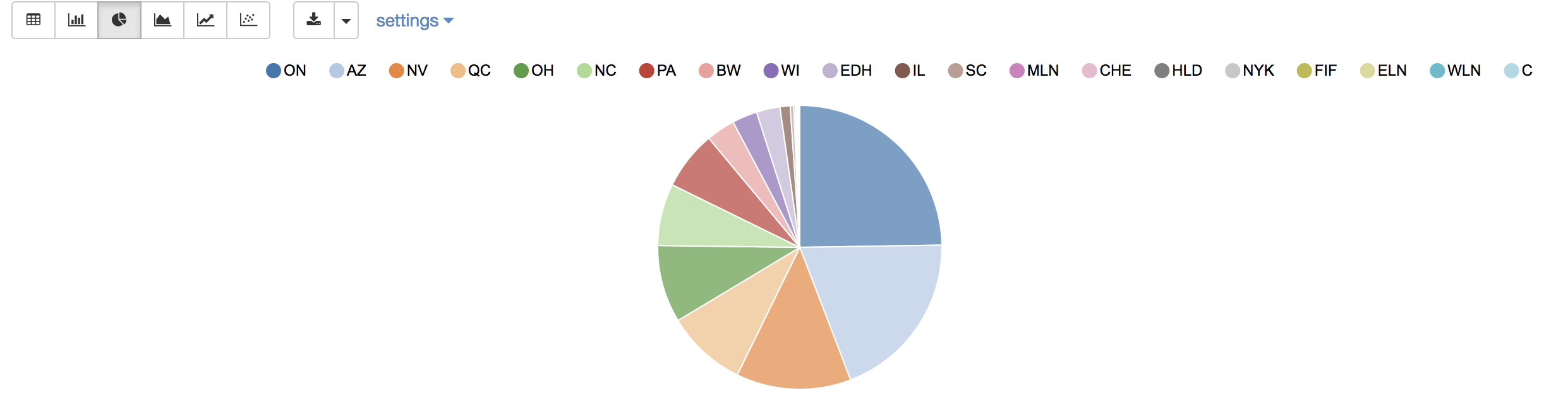
|  |
| --- |
| // review of restaurants val revRestDF = spark.sql("""  SELECT restaurants\_df.business\_id, review\_df.stars, review\_df.user\_id  FROM review\_df JOIN restaurants\_df ON restaurants\_df.business\_id = review\_df.business\_id  """) revRestDF.show  val review\_filteredDF = reviewDF.**join**(restaurantsDF, reviewDF("business\_id") === restaurantsDF("business\_id")).  **select**(restaurantsDF("business\_id"),reviewDF("stars"), reviewDF("user\_id") ) review\_filteredDF.show  // whichever way you make revRestDF, you must create  // a temp view to use in SQL statements revRestDF.createTempView("revRest\_df") |

Create Elite users.

|  |
| --- |
| //create elite users. Use SQL **as** well **as** column expression val eliteDF = spark.sql("""   SELECT \* FROM user\_df LATERAL VIEW explode(elite) c AS elite\_year  """) eliteDF.show eliteDF.createTempView("elite\_df") val elite2DF = userDF.withColumn("elite\_year", explode($"elite")) elite2DF.show |

## 1) Restaurants across United States

|  |
| --- |
| // number of restaurants in the US  %sql **SELECT** state, **count** (business\_id) number\_restaurants **FROM** restaurants\_df **GROUP** **BY** state **ORDER** **BY** number\_restaurants **DESC** **LIMIT** 20 |

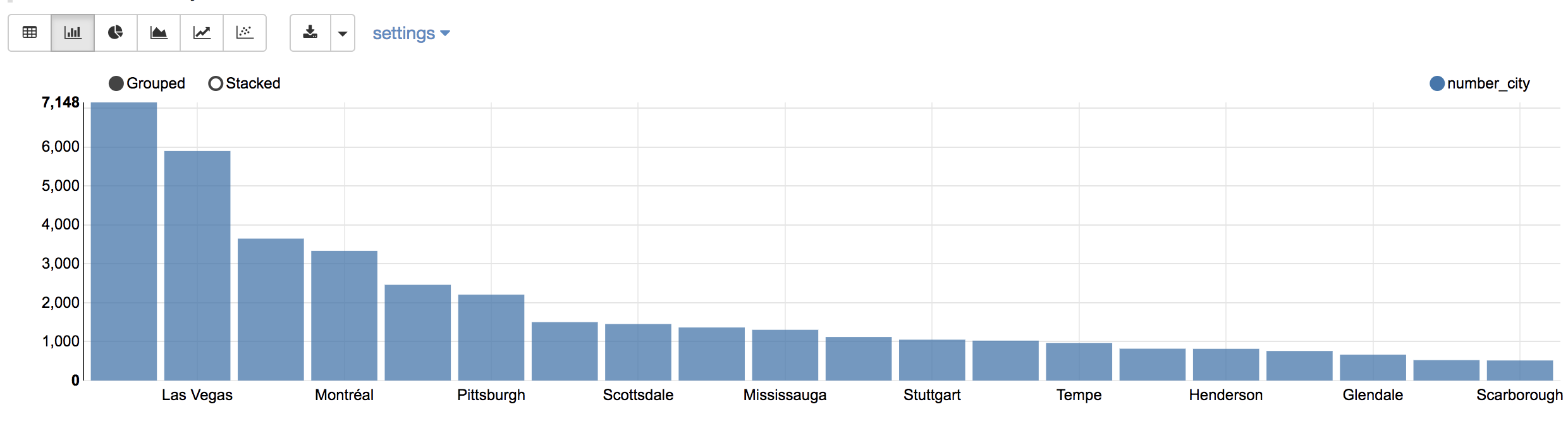


|  |
| --- |
| // column expression val USRestaurantsDF = restaurantsDF.  groupBy("state").  **count**().withColumnRenamed("count", "num\_restaurants").  orderBy(**desc**("num\_restaurants")) USRestaurantsDF.show |

2) Which Cities Have The Highest Number of Restaurants? In order to map restaurants across United, Select the columns (**city**, count of **business\_id** as **number\_city** ) by grouping data of restaurants table by **city** and then order it by **number\_city**

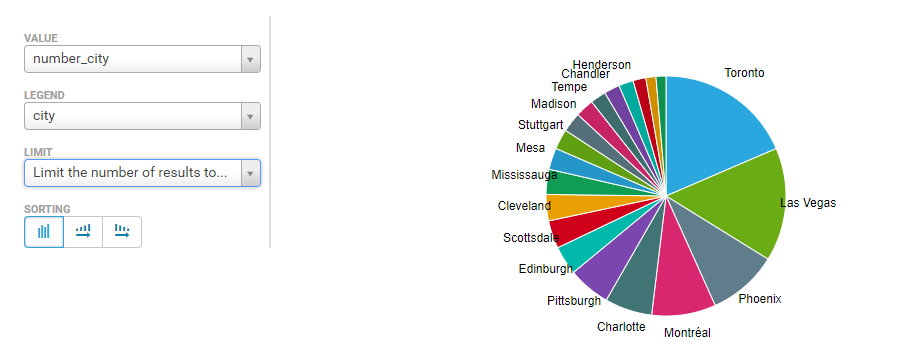
# http://allaboutscala.com/big-data/spark/#dataframe-sql-group-by

|  |
| --- |
| %sql **SELECT** city, **count**(business\_id) number\_city **FROM** restaurants\_df **GROUP** **BY** city **ORDER** **BY** number\_city **DESC** **LIMIT** 20; |



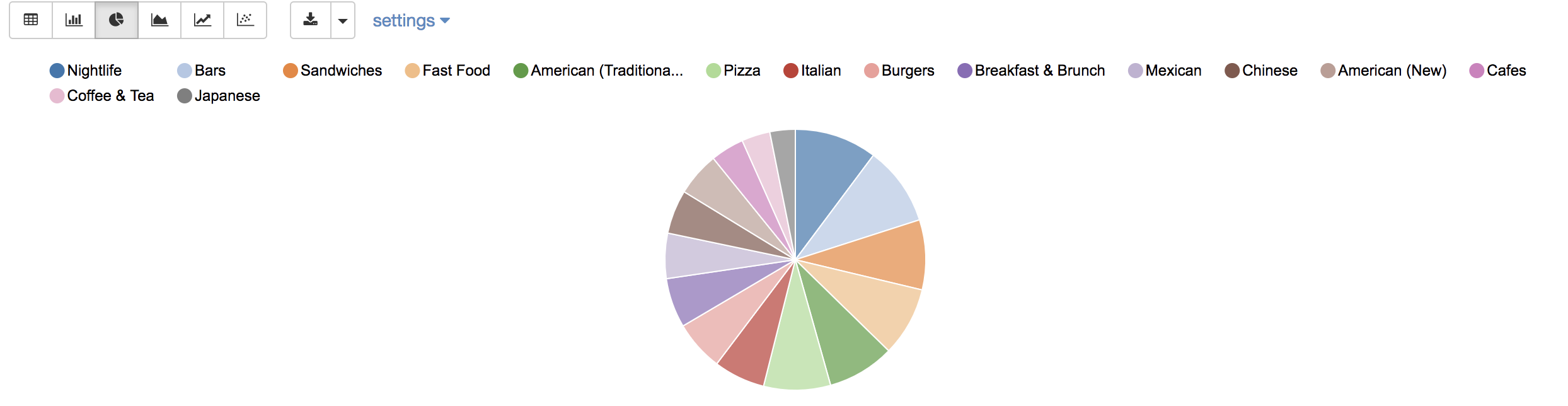
|  |
| --- |
| // column expression val city\_restDF = restaurantsDF.  groupBy("city").  **count**().withColumnRenamed("count","num\_city").  orderBy(**desc**("num\_city")) city\_restDF.show() |

Below is the pie chart version you created in Hue.



3) Find out Top 15 Sub-Categories Of Restaurants from tables **exploded**  and **restaurants** with **business\_id** and grouping by **cat\_exploded** column of the table **exploded,** which are not in("Restaurants","Food")

|  |
| --- |
| %sql **SELECT** e.cat\_exploded **category**, **count**(e.cat\_exploded) number **FROM** exploded\_df e **JOIN** restaurants\_df re **ON** e.business\_id = re.business\_id **WHERE** e.cat\_exploded **not** **in** ("Restaurants","Food") **GROUP** **BY** e.cat\_exploded **ORDER** **BY** number **DESC** **LIMIT** 15 |



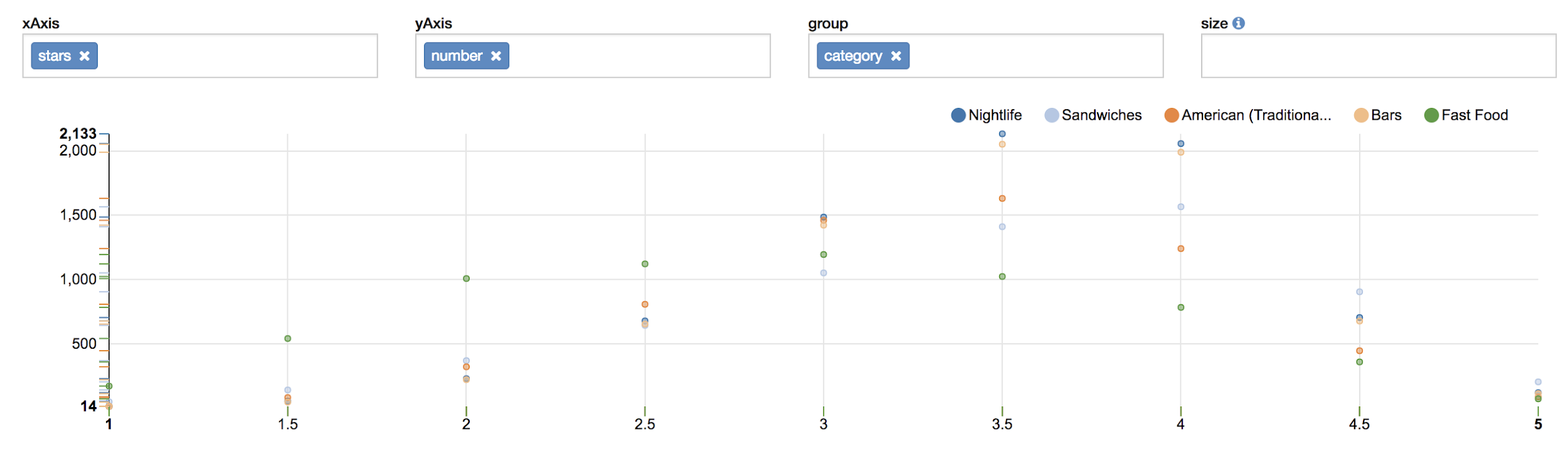
|  |
| --- |
| // column expression val t15JoinDF = explodedDF.  join(restaurantsDF, explodedDF("business\_id") === restaurantsDF("business\_id")) val t15JoinFilterDF = t15JoinDF.  filter(!explodedDF("cat\_exploded").isin("Restaurants","Food")).  select(explodedDF("cat\_exploded")) val t15groupDF = t15JoinFilterDF.  groupBy(explodedDF("cat\_exploded")).  count().withColumnRenamed("count","num\_subcat").  orderBy(desc("num\_subcat")).limit(20) t15groupDF.show()  // or chaining val t15JoinDF = explodedDF.  join(restaurantsDF, explodedDF("business\_id") === restaurantsDF("business\_id")).  filter(!explodedDF("cat\_exploded").isin("Restaurants","Food")).  select(explodedDF("cat\_exploded")).  groupBy(explodedDF("cat\_exploded")).  count().withColumnRenamed("count","num\_subcat").  orderBy(desc("num\_subcat")).limit(20) t15JoinDF.show() |

4) Distribution of ratings vs categories:

테이블 ratings를 앞서 만든 테이블 exploded 와 restaurant을 가지고 같은 business\_id로 조인하여 다음의 조건을 만족하여 만드세요:

* exploded 의 cat\_exploded 값은 다음의 값중 하나: "Nightlife","Bars", "Sandwiches", "Fast Food","American (Traditional)"
* exploded 의 cat\_exploded 와 stars 컬럼을 가지고 Group By 하고 stars 컬럼으로 ASC 하여 정렬하세여;
* 다음의 HiveQL로 Hue에서 그래프를 다음과 같이 만듬: select \* from ratings order by number DESC;

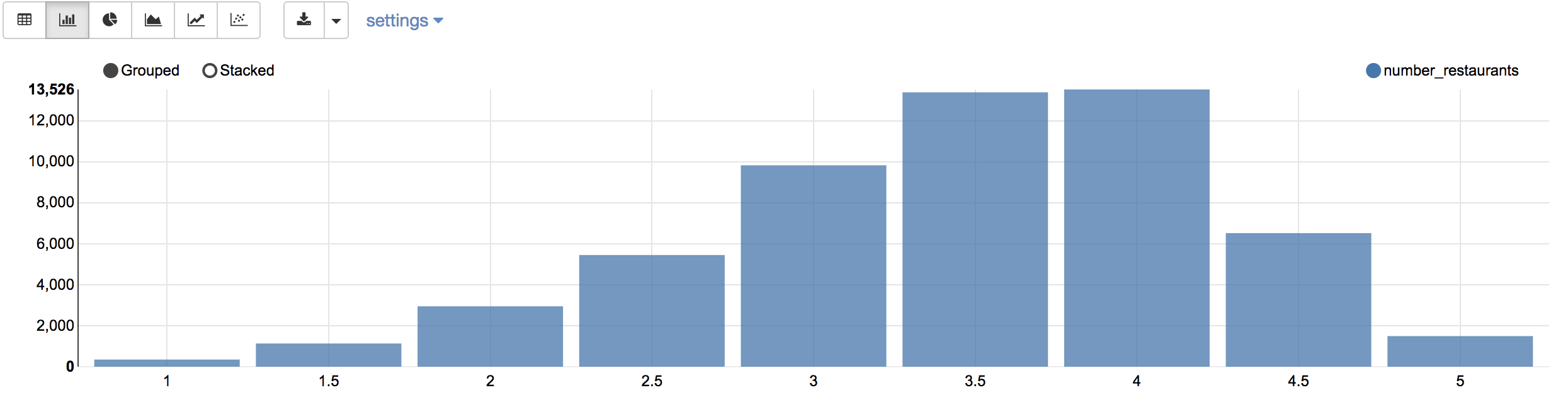
|  |
| --- |
| %sql **SELECT** e.cat\_exploded **category**, e.stars stars, **count**(e.cat\_exploded) number  **FROM** exploded\_df e **JOIN** restaurants\_df re **ON** e.business\_id = re.business\_id **WHERE** e.cat\_exploded **in** ("Nightlife","Bars","Sandwiches","Fast Food","American (Traditional)") **GROUP** **BY** e.cat\_exploded, e.stars **ORDER** **BY** stars **ASC** |



|  |
| --- |
| // column expression val rateCatDF = explodedDF.  join(restaurantsDF, explodedDF("business\_id") === restaurantsDF("business\_id")).  filter(explodedDF("cat\_exploded").isin("Nightlife","Bars","Sandwiches","Fast Food","American (Traditional)")).  select(explodedDF("cat\_exploded"), explodedDF("stars")).  groupBy(explodedDF("cat\_exploded"), explodedDF("stars")).   count().withColumnRenamed("count","number").  orderBy("stars", "number") rateCatDF.show() |

5) What ratings do the majority of restaurants have?

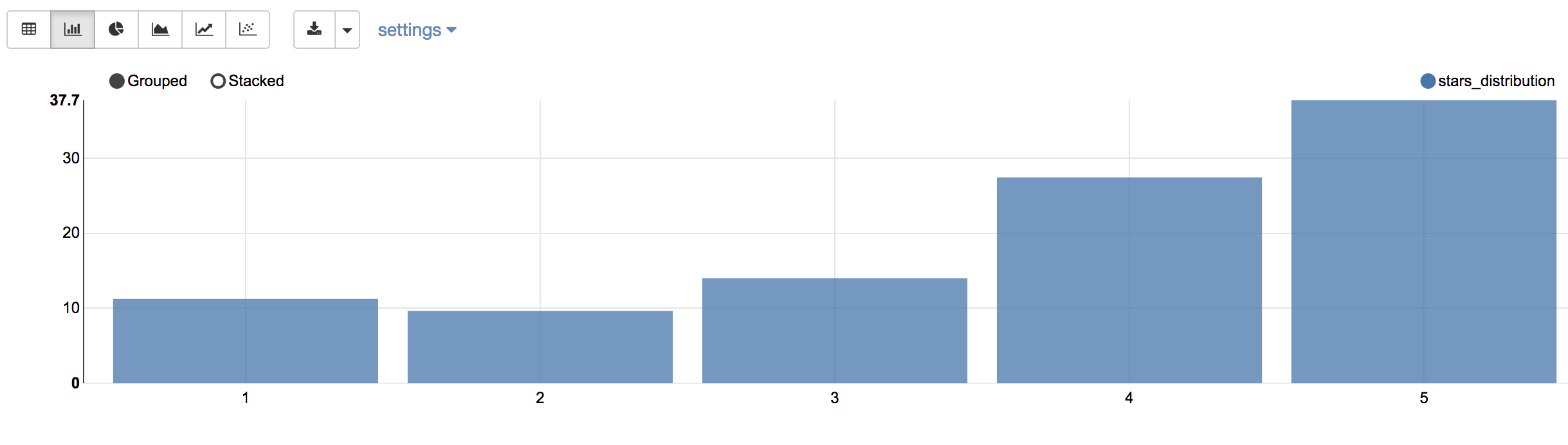
|  |
| --- |
| %sql **SELECT** stars, **count** (business\_id) number\_restaurants  **FROM** restaurants\_df  **GROUP** **BY** stars  **ORDER** **BY** stars |



|  |
| --- |
| // column expression val rRatingsDF = restaurantsDF.**select**("stars").  groupBy("stars").  count().withColumnRenamed("count", "num\_restaurants") rRatingsDF.show() |

6) Rating distribution in restaurant reviews

|  |
| --- |
| %sql **SELECT** stars,   **round**(**count**(stars) \* 100.0 / **sum**(**count**(stars)) **over**(),2) stars\_distribution   **FROM** revRest\_df  **GROUP** **BY** stars   **ORDER** **BY** stars |



# not showing the decimal number in text of HDFS

CREATE TABLE stars2 AS  
SELECT stars, round((count(stars) \* 100.0 / sum(count(stars)) over()), 2) stars\_distribution   
FROM review\_filtered  
GROUP BY stars;

# test

SELECT stars, count(stars), sum(count(stars)) over()  
FROM review\_filtered  
GROUP BY stars;

# not showing the decimal number in text of HDFS

CREATE TABLE stars2

AS  
SELECT stars, [FIll-In] stars\_distribution   
FROM review\_filtered  
GROUP BY stars;

FIl-In:

round((a \* 100.0 / b), 2)

a. count(stars)

b. a의 sum over 함수

# not showing the decimal number in Json of HDFS

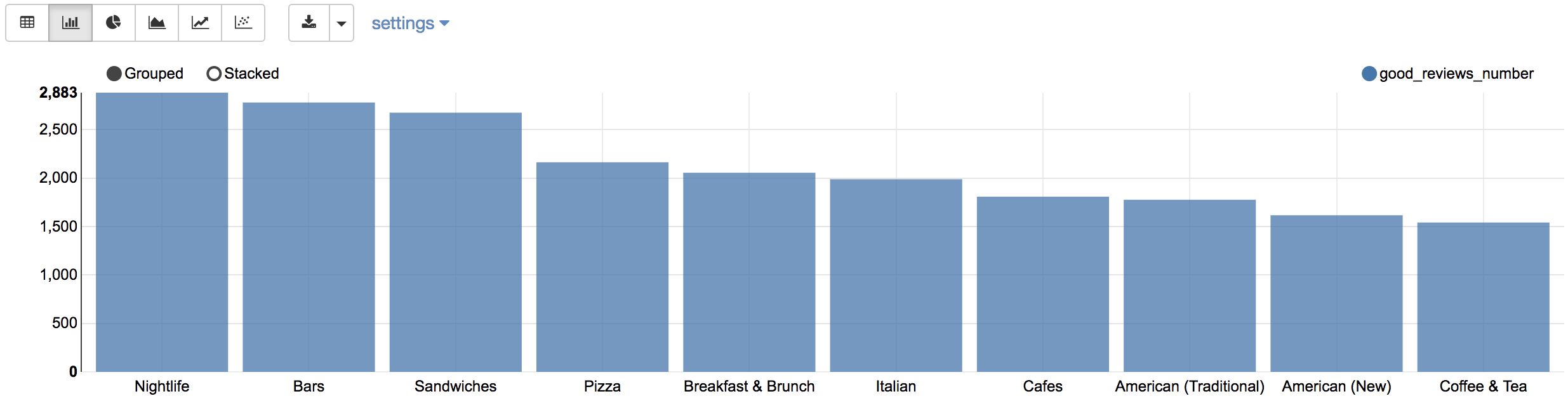
CREATE TABLE stars  
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe' AS  
SELECT stars, round((count(stars) \* 100.0 / sum(count(stars)) over()), 2) stars\_distribution   
FROM review\_filtered  
GROUP BY stars;

SELECT \* FROM stars2 ORDER BY stars\_distribution ;

7) Which restaurants get bad vs good reviews?

1. Good Review

|  |
| --- |
| %sql **SELECT** e.cat\_exploded **category**,   **count**(e.cat\_exploded) good\_reviews\_number  **FROM** exploded\_df e **JOIN** restaurants\_df re **ON** e.business\_id = re.business\_id **WHERE** e.cat\_exploded **NOT** **IN** ("Restaurants","Food") **AND** re.stars>=4  **GROUP** **BY** e.cat\_exploded **ORDER** **BY** good\_reviews\_number **DESC** **LIMIT** 10 |



**# 힌트:**

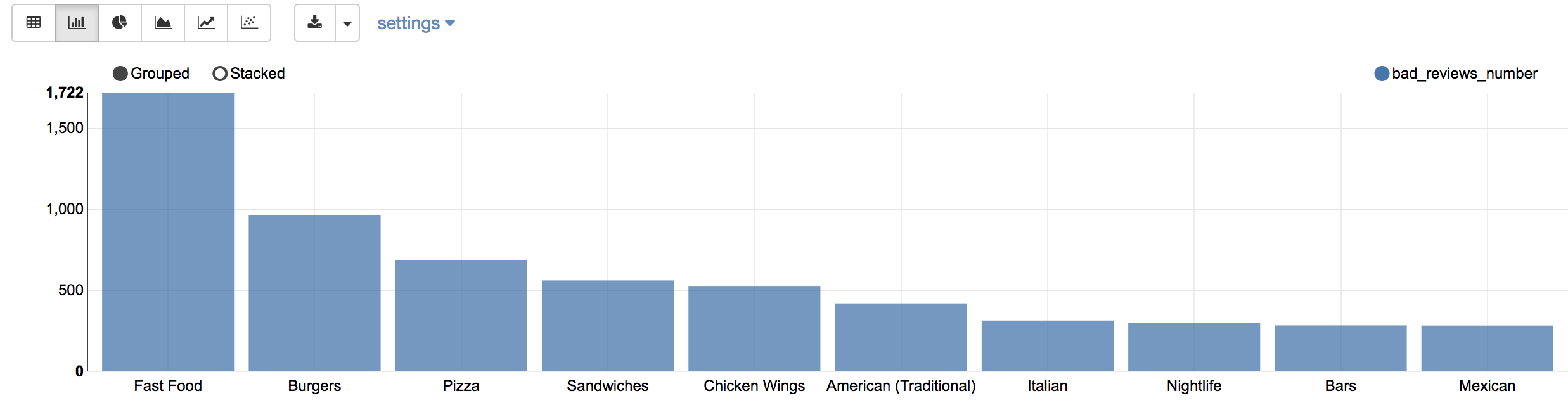
good\_reviews1 테이블을 exploded e 와 restaurants re 테이블을 business\_id로 조인하여 컬럼은 (e.cat\_exploded category, count(e.cat\_exploded) good\_reviews\_number) 을 가지고 다음의 조건을 만족하게 만드세요:  
- e.cat\_exploded NOT IN ("Restaurants","Food")

- re.stars>=4

|  |
| --- |
| //column expression val goodRevDF = explodedDF.  join(restaurantsDF, explodedDF("business\_id") === restaurantsDF("business\_id")).  filter(!explodedDF("cat\_exploded").isin("Restaurants","Food") && (restaurantsDF("stars") >= 4) ).  select(explodedDF("cat\_exploded")).  groupBy(explodedDF("cat\_exploded")).  count().withColumnRenamed("count","num\_good").  orderBy(desc("num\_good")).limit(10) goodRevDF.show() |

1. Bad Review

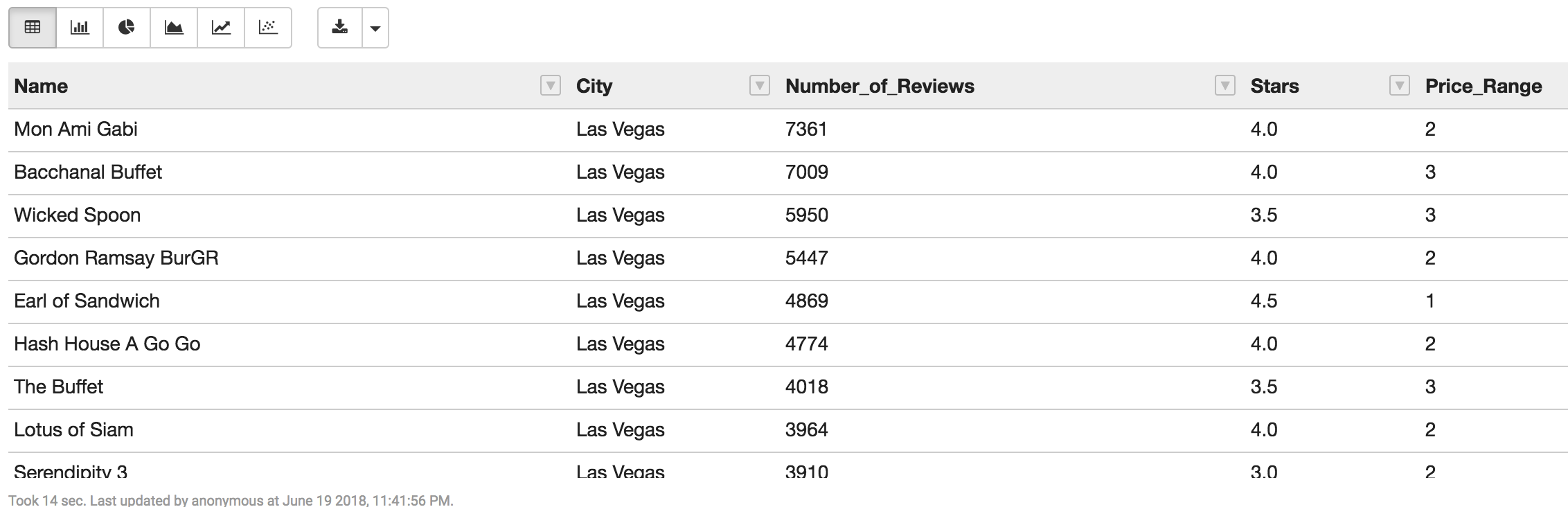
|  |
| --- |
| %sql **SELECT** e.cat\_exploded **category**, **count**(e.cat\_exploded) bad\_reviews\_number  **FROM** exploded\_df e **JOIN** restaurants\_df re **ON** e.business\_id = re.business\_id **WHERE** e.cat\_exploded **NOT** **IN** ("Restaurants","Food")   **AND** re.stars<=2 **GROUP** **BY** e.cat\_exploded **ORDER** **BY** bad\_reviews\_number **DESC** **LIMIT** 10 |



|  |
| --- |
| // column expression val badRevDF = explodedDF.  join(restaurantsDF, explodedDF("business\_id") === restaurantsDF("business\_id")).  filter(!explodedDF("cat\_exploded").isin("Restaurants","Food") && (restaurantsDF("stars") <= 2) ).  select(explodedDF("cat\_exploded")).  groupBy(explodedDF("cat\_exploded")).  count().withColumnRenamed("count","num\_bad").  orderBy(desc("num\_bad")).limit(10) badRevDF.show() |

8) Which restaurants have the most reviews?

|  |
| --- |
| %sql **SELECT** **name** **Name**, city City, review\_count Number\_of\_Reviews ,  stars Stars, attributes.restaurantspricerange2 Price\_Range **FROM** restaurants\_df **ORDER** **BY** review\_count **DESC** **LIMIT** 15 |



|  |
| --- |
| // column expression val mostReviewDF = restaurantsDF.  **select**($"name".**alias**("Name"), $"city".**alias**("City"),   $"review\_count".**alias**("Number\_of\_Reviews"),   $"stars".**alias**("Stars"),   $"attributes.restaurantspricerange2".**alias**("Price\_Range")).  orderBy(desc("Number\_of\_Reviews")).  limit(15) mostReviewDF.show() |

9) What number of yelp users are elite? Do they rate differently than non-elite users?

1. Average rating by all users:

|  |
| --- |
| %sql **SELECT** **round**(**avg**(average\_stars),2) avg\_rating\_user **FROM** user\_df |

3.71

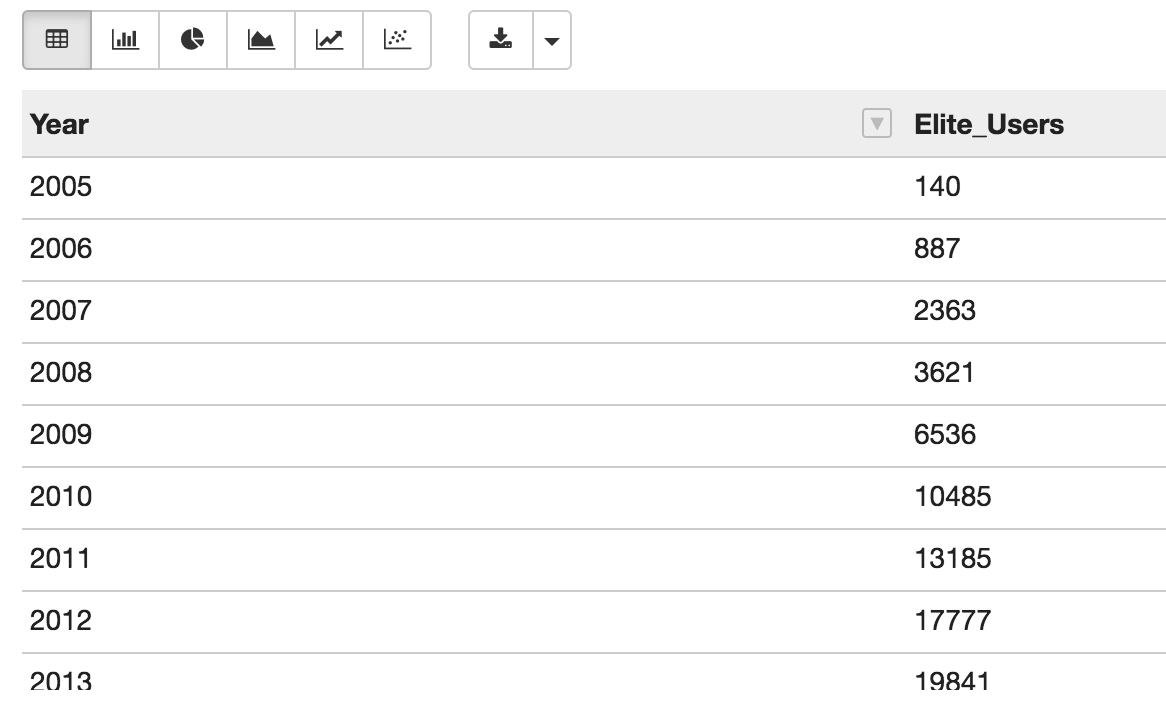
1. Average rating by elite users:

|  |
| --- |
| %sql **SELECT** **round**(**avg**(average\_stars),2) avg\_rating\_elite **FROM** elite\_df |

3.82

1. Count number of elite users by year:

|  |
| --- |
| %sql  SELECT elite\_year Year, count(distinct user\_id) Elite\_Users FROM elite\_df  GROUP BY elite\_year  ORDER BY elite\_year ASC |



1. Count average reviews by elite users by year

|  |
| --- |
| %sql **SELECT** elite\_year **Year**, **round**(**avg**(average\_stars),2) Avg\_Rating  **FROM** elite\_df **GROUP** **BY** **Year** **ORDER** **BY** Avg\_Rating **ASC** |



Try creating the column expression versions by yourself.