



Term Project Report

On

Yelp Reviews Analysis

**CIS8040, Unstructured Data Management
Fall 2017**

- Shagun Garg
- Ashish Devrani
- Sanchit Bogra
- Jasdev Singh Sachdeva

Pseudo schema for MongoDB

Yelp is a go to application for any sort of outing be it a restaurant, lounge, club or a tourist spot. However, we have tried to incorporate some of our own ideas to provide an enhanced and fun experience to the user when they are using the Yelp application. The starting point of our proposed enhancements, is the Yelp Schema.

1. We have noticed that, in the Yelp schema, a few variables, although take only Boolean values (0,1), are stored as integer. To improve the performance of this database, we could store these variables as Boolean values (true, false), which would be more memory efficient.
2. Secondly, we have incorporated another collection comment in the pseudo-schema. The basic idea behind this suggestion is that the presence of comments, is beneficial for improving the review quality. Under such pressure the reviewers are more focused to perform better as others can comment on their performance, which might question their view or approach. This very pressure to maintain their reputation pushes the reviewers to perform better and improves the overall quality of the review system .

```
{
  business:[
    {
      "business_id":"YDf95gJZaq05wvo7hTQbbQ",
      "name":"Richmond Town Square",
      "neighborhood":"",
      "address":"691 Richmond Rd",
      "city":"Richmond Heights",
      "state":"OH",
      "postal_code":"44143",
      "latitude":41.5417162,
      "longitude":-81.4931165,
      "stars":2.0,
      "review_count":17,
      "is_open":true,
      "attributes":{
        "RestaurantsPriceRange2":2,
        "BusinessParking":{
          "garage":false,
          "street":false,
          "validated":false,
          "lot":true,
          "valet":false
        },
        "BikeParking":true,
        "PokestopNearby": false,
        "WheelchairAccessible":true
      },
      "categories":[
        "Shopping",
        "Shopping Centers"
      ],
      "hours":{
        "Monday":"10:00-21:00",
        "Tuesday":"10:00-21:00",
        "Friday":"10:00-21:00",
        "Wednesday":"10:00-21:00",
        "Thursday":"10:00-21:00",

```

```
"Sunday":"11:00-18:00",
"Saturday":"10:00-21:00"
}
},
{
  "business_id":"mLwM-h2YhXl2NCgdS84_Bw",
  "name":"South Florida Style Chicken & Ribs",
  "neighborhood":"Eastland",
  "address":"2824 Milton Rd",
  "city":"Charlotte",
  "state":"NC",
  "postal_code":"28215",
  "latitude":35.23687,
  "longitude":-80.7419759,
  "stars":4.5,
  "review_count":4,
  "is_open":false,
  "attributes":{
    "GoodForMeal":{
      "dessert":false,
      "latenight":false,
      "lunch":false,
      "dinner":false,
      "breakfast":false,
      "brunch":false
    },
    "HasTV":false,
    "RestaurantsGoodForGroups":true,
    "NoiseLevel":"average",
    "RestaurantsAttire":"casual",
    "RestaurantsReservations":false,
    "OutdoorSeating":false,
    "BusinessAcceptsCreditCards":false,
    "RestaurantsPriceRange2":2,
    "RestaurantsDelivery":true,
    "PokestopNearby": true,
    "Ambience":{
      "romantic":false,
      "intimate":false,
      "classy":false,
      "hipster":false,
      "divey":false,
      "touristy":false,
      "trendy":false,
      "upscale":false,
      "casual":false
    },
    "RestaurantsTakeOut":true,
    "GoodForKids":true
  },
  "categories":[
    "Food",
    "Soul Food",
    "Convenience Stores",
    "Restaurants"
```

```

],
"hours":{
  "Monday":"10:00-22:00",
  "Tuesday":"10:00-22:00",
  "Friday":"10:00-22:00",
  "Wednesday":"10:00-22:00",
  "Thursday":"10:00-22:00",
  "Sunday":"10:00-22:00",
  "Saturday":"10:00-22:00"
}
}
],

```

```

review:[
  {
    "review_id":"i5UwUPIQFPLcE8p2gPFwBw",
    "user_id":"WZXp9-V2dqRRJqhGgRqueA",
    "business_id":"jQsNFOzDpxPmOurSWCg1vQ",
    "stars":4,
    "date":"2015-03-26",
    "text":"For being fairly "      fast" food.. Pei Wei (pronounced pay way I confirmed...",
    "useful":1,
    "funny":0,
    "cool":0,
    "review_comments": [
      "VZXDC7VBdIXXjE3omVqfdfr",
      "ABCDC7VBdIrfjE3omVqfdfr"
    ]
  },
  {
    "review_id":"ne5WhI1jUFOcRn-b-gAzHA",
    "user_id":"AXgRULmWcME7J6Ix3I--ww",
    "business_id":"uYHaNptLzDLoV_JZ_MuzUA",
    "stars":3,
    "date":"2015-09-17",
    "text":"Mittlerweile gibt es in Edinburgh zwei Ableger der Motel-One-Kette - d...",
    "useful":0,
    "funny":0,
    "cool":0,
    "review_comments": [
      "KZXDC7VBdIXXjE3omVqfdfr",
      "QRCDC7VBdIrfjE3omVqfdfr"
    ]
  }
],

```

```

user:[
  {
    "user_id":"YHJIMK_zVH_VY6HCY6bYvg",
    "name":"Erica",
    "review_count":68,
    "yelping_since":"2012-06-08",
    "friends":[

```

```

        "Puvuej6lzJ1JOEmtjG7V_Q",
        "fq7CL1myWPYeH0d4bKtsIw",
        "rAsJznwttFfuxNEe7bTGFA",
        "a-Ug_MFryz3utca-NaMkNQ"
    ],
    "useful":5,
    "funny":3,
    "cool":1,
    "fans":4,
    "elite":[
        2015,
        2016
    ],
    "average_stars":4.06,
    "compliment_hot":5,
    "compliment_more":5,
    "compliment_profile":1,
    "compliment_cute":0,
    "compliment_list":0,
    "compliment_note":4,
    "compliment_plain":5,
    "compliment_cool":4,
    "compliment_funny":4,
    "compliment_writer":22,
    "compliment_photos":5,
    "badges": [
        "Rookie",
        "Barfly",
        "Footloose"
    ],
    "location": "Atlanta, GA "
},
{
    "user_id":"PcvbBOCOcs6_suRDH7TSTg",
    "name":"Juan",
    "review_count":921,
    "yelping_since":"2012-08-16",
    "friends":[
        "iN0A6QIrEFYoSGHFaknh8Q",
        "B2HDoWNlzlLlon0lhS1cmDw",
        "sEYD5YG0fGgvLXAZ5RDuZg",
        "3R_dB9VQ_D3WPJEw7pmorA"
    ],
    "useful":9152,
    "funny":360,
    "cool":6006,
    "fans":49,
    "elite":[
        2017,
        2016
    ],
    "average_stars":4.23,
    "compliment_hot":59,
    "compliment_more":8,
    "compliment_profile":3,

```

```
"compliment_cute":0,
"compliment_list":0,
"compliment_note":51,
"compliment_plain":386,
"compliment_cool":276,
"compliment_funny":276,
"compliment_writer":29,
"compliment_photos":169,
  "badges": [
    "VIP",
    "Crowd Puller",
    "Wingman"
  ],
  "location": "Brooklyn, NY"
}
],
```

```
checkin:[
  {
    "time":{
      "Thursday":{
        "21:00":4,
        "1:00":1,
        "4:00":1,
        "2:00":1,
        "20:00":2,
        "22:00":1,
        "19:00":1,
        "15:00":2,
        "13:00":1,
        "23:00":2
      },
      "Wednesday":{
        "11:00":2,
        "13:00":2,
        "14:00":1,
        "17:00":1,
        "6:00":1,
        "2:00":1,
        "0:00":2,
        "1:00":1,
        "21:00":1,
        "18:00":1,
        "19:00":1,
        "20:00":2
      },
      "Sunday":{
        "18:00":1,
        "16:00":1,
        "14:00":1,
        "19:00":2,
        "17:00":1,
        "23:00":1,
        "21:00":1,
        "20:00":5,
```

```
"6:00":1,
"0:00":1,
"2:00":2,
"3:00":3
},
"Friday":{
  "16:00":1,
  "14:00":2,
  "10:00":2,
  "23:00":1,
  "19:00":2,
  "18:00":1,
  "15:00":1,
  "21:00":2,
  "22:00":2,
  "3:00":1,
  "0:00":2
},
"Saturday":{
  "21:00":1,
  "23:00":3,
  "18:00":4,
  "10:00":1,
  "12:00":1,
  "13:00":3,
  "14:00":1,
  "15:00":1,
  "16:00":2,
  "17:00":3,
  "2:00":1,
  "0:00":1,
  "1:00":2
},
"Monday":{
  "12:00":1,
  "11:00":1,
  "14:00":1,
  "18:00":1,
  "19:00":1,
  "23:00":1,
  "20:00":1
},
"Tuesday":{
  "18:00":2,
  "12:00":1,
  "13:00":2,
  "16:00":1,
  "15:00":1,
  "4:00":1,
  "21:00":1,
  "20:00":2,
  "23:00":2
}
},
"business_id":"7KPBkxAOEt3QeIL9PEErg"
```

```

    },
    {
      "time":{
        "Monday":{
          "13:00":1
        },
        "Thursday":{
          "20:00":1,
          "13:00":1
        },
        "Sunday":{
          "19:00":1
        },
        "Wednesday":{
          "17:00":1
        },
        "Saturday":{
          "21:00":1,
          "16:00":1
        }
      },
      "business_id":"kREVIrSBbtqBhIYkTccQUg"
    }
  ],
  photos:[
    {
      "photo_id":"VZXDC7VBdIXXjE3omVqeMg",
      "business_id":"JzB7NITHQ7gVHGVZ1ntgIQ",
      "caption":"Black Angus Steak Sandwich... Huge!!!",
      "label":"food"
    },
    {
      "photo_id":"c6Em6dDZ4aVKDI8Lc2BQog",
      "business_id":"JzB7NITHQ7gVHGVZ1ntgIQ",
      "caption":"",
      "label":"outside"
    }
  ],
  tip:[
    {
      "text":"Get here early enough to have dinner.",
      "date":"2012-07-15",
      "likes":0,
      "business_id":"tJRDl15yqpZwehenzE2cSg",
      "user_id":"zcTZk7OG8ovAmh_fenH21g"
    },
    {
      "text":"Great breakfast large portions and friendly waitress. I highly recommend it",
      "date":"2015-08-12",
      "likes":0,
      "business_id":"jH19V2I9fIslNhdzPmdkA",
      "user_id":"ZcLKXikTHYOnYt5VYRO5sg"
    }
  ],

```



```
comment: [  
  {  
    "comment_id": "VZXDC7VBdIXXjE3omVqfdfr",  
    "text": "I don't agree with the reviewer at all. My experience was awesome",  
    "user_id": "zcTZk7OG8ovAmh_fenH21g",  
    "review_id": "ne5WhI1jUFOcRn-b-gAzHA"  
  },  
  {  
    "comment_id": "VrwwDC7VBdIXXjE3omVqfdfr",  
    "text": "Thanks for writing this",  
    "user_id": "zcTZk7OG8ovAmgrgfenH21g",  
    "review_id": "ne5WhI1jUFOcRn-rfeAzHA"  
  }  
]  
}]
```

Properties of MongoDB Pseudo Schema

- What databases:
 - **NoSQL – MongoDB** – We are using this database because we are working with semistructured/unstructured data. Instead of storing data in tables, it stores data in
 - collections made of individual documents.
- Collections:
 - Grouping of MongoDB documents is called Collection. Here, we are using 7 collections namely business, review, user, checkin, photos, tip, comment. In each collection there are various documents that have been created.
- Documents:

Business

- For the Business collection, we have documents such as business_id, name, neighborhood, address, city, state, postal_code, latitude, longitude, stars, review_count, is_open, attributes, categories, hours.
- The attributes document further gets divided into sub-documents like BikeParking, WheelchairAccessible, RestaurantPriceRange, etc.
- We have added a new attribute sub-document to the original schema called PokestopNearby to connect the users with the new trend of Pokemon Go that has gone viral in recent times.
- The categories document further gets divided into sub-documents like Shopping, Restaurants, Theme Parks, etc.

Review

- For the Review collection, we have documents such as review_id, business_id, user_id, stars, date, text, useful, funny, cool, review_comments.
- These documents provide a reference with collections namely Business and User and a new collection Comment which we will discuss further later.

User

- For the User collection, we have documents such as user_id, name, review_count, yelping_since, friends, useful, funny, cool, fans, elite, average_stars, compliment_hot, compliment_more, compliment_profile, compliment_cute, compliment_list, compliment_note, compliment_plain, compliment_cool, compliment_funny, compliment_writer, compliment_photos, badges, location.
- The documents badges and location have been added by us to provide a better understanding of the user's profile. This is in relation to the Amazon dataset that we were exposed to which leads a user towards more helpfulness of a review once they see badges attached to the reviewer's profile along with the city they reside in.

Checkin

- For the Checkin collection, we have documents such as time and business_id.
- The time document is further divided into sub-documents including the day of the week and time like Monday, Tuesday, 13:00, 12:00, etc.

Photos

- For the Photos collection, we have documents such as photo_id, business_id, caption, label.
- Here we provide a reference with the Business collection which we will discuss further later.

Tip

- For the Tip collection, we have documents such as text, date, likes, business_id, user_id.
- These documents provide a reference with both collections namely Business and User which we will discuss further later.

Comment

- For the Comment collection, we have documents such as comment_id, text, user_id, review_id.
- These documents provide a reference with both collections namely Review and User which we will discuss further later.

➤ Referencing and Embedding:

- For the Business collection – we are embedding every document in the Business collection.
- For the Review collection – The attributes business_id and user_id are referenced in the Review collection.
- For the User collection – we are embedding every document in the User collection.
- For the Checkin collection – The attribute business_id is being referenced in the checkin collection.
- For the Photos collection – The attribute business_id is being referenced in the photos collection.
- For the Tip collection – The attributes business_id and user_id are referenced in the tip collection.
- For the Comment collection – The attributes review_id and user_id are referenced in the comment collection.

➤ Indexing:

- Indexing supports efficient retrieval of data by execution of queries. There are multiple ways by which we can do indexing. By default, all collections have an index on the _id field, and we may add additional indexes to support important queries and operations.
- Another way of indexing is Single-Field Indexing.
- For the collections business, review, user and photos, we are over-riding the Default indexing and creating single field index on _ID attribute in all our collections like business_id, user_id, review_id, photo_id.
- For the collections checkin and tip, a default indexing of _id is used which is not shown in our schema.

➤ Constraints imposed:

For our schema, we have included constraints contained in the below attached image by taking reference from <https://docs.mongodb.com/v3.0/reference/bsontypes/>.

Document	Attribute	Data Format	MongoDB Data Type	Data Change	Constraint
Business	Business_ID	Text	String	Static	Unique, NotNull
	Name	Text	String	Static	Not Null
	Neighborhood	Text	String	Static	Can be Null
	Address	Text	String	Static	Not Null, Positive Value
	City	Numeric	Double	Static	Not Null, Positive Value
	State	Numeric	Double	Static	Not Null, Positive Value
	Postal_Code	Text	Array[String]	Static	Not Null, Positive Value
	Latitude	Numeric	Double	Static	Not Null
	Longitude	Numeric	Double	Static	Not Null
	Stars	Numeric	Double	Dynamic	Can be Null, Positive Value
	Review_Count	Numeric	Double	Dynamic	Can be Null, Positive Value
	Is_Open	Boolean	Boolean	Dynamic	Not Null
	Attributes	Text	Array[String]	Dynamic	Not Null
	Categories	Text	Array[String]	Static	Not Null
	Hours	Text	Array[String]	Static	Not Null

Document	Attribute	Data Format	MongoDB Data Type	Data Change	Constraint
Review	Review_ID	Text	String	Static	Unique, NotNull
	User_ID	Text	String	Static	Not Null
	Business_ID	Text	String	Static	Not Null
	Stars	Numeric	String	Static	Can be Null, Positive Value
	Date	DateTime	Date	Static	Not Null, InRange(1-5)
	Text	Text	String	Static	Not Null
	Useful	Numeric	Double	Dynamic	Can be Null, Positive Value
	Funny	Numeric	Double	Dynamic	Can be Null, Positive Value
	Cool	Numeric	Double	Dynamic	Can be Null, Positive Value
	Review_Comments	Text	Array[String]	Dynamic	Can be Null

Document	Attribute	Data Format	MongoDB Data Type	Data Change	Constraint
User	User_ID	Text	String	Static	Unique, NotNull
	Name	Text	String	Static	Not Null
	Review_Count	Numeric	Double	Dynamic	Can be Null, Positive Value
	Yelping_Since	DateTime	Date	Static	Not Null
	Friends	Text	Array[String]	Dynamic	Can be Null, Positive Value
	Useful	Numeric	Double	Dynamic	Can be Null, Positive Value
	Funny	Numeric	Double	Dynamic	Can be Null, Positive Value
	Cool	Numeric	Double	Dynamic	Can be Null, Positive Value
	Fans	Numeric	Double	Dynamic	Can be Null, Positive Value
	Elite	Numeric	Array[Number]	Dynamic	Can be Null, Positive Value
	Average_Stars	Numeric	Double	Dynamic	Can be Null, Positive Value
	Compliment(Various)	Numeric	Double	Dynamic	Can be Null, Positive Value
	Badges	Text	Array[String]	Dynamic	In List of Badges
	Location	Text	String	Dynamic	Not Null

Document	Attribute	Data Format	MongoDB Data Type	Data Change	Constraint
Checkin	Time	DateTime	Array[Day/Time]	Static	NotNull
	Business_ID	Text	String	Static	Not Null

Document	Attribute	Data Format	MongoDB Data Type	Data Change	Constraint
Photos	Photo_ID	Text	String	Static	Unique, NotNull
	Business_ID	Text	String	Static	Not Null
	Caption	Text	String	Static	Can be Null
	Label	Text	String	Static	Can be Null

Document	Attribute	Data Format	MongoDB Data Type	Data Change	Constraint
Tip	Text	Text	String	Static	NotNull
	Date	DateTime	Date	Static	Not Null
	Likes	Numeric	Double	Dynamic	Can be Null
	Business_ID	Text	String	Static	Not Null
	User_ID	Text	String	Static	Not Null

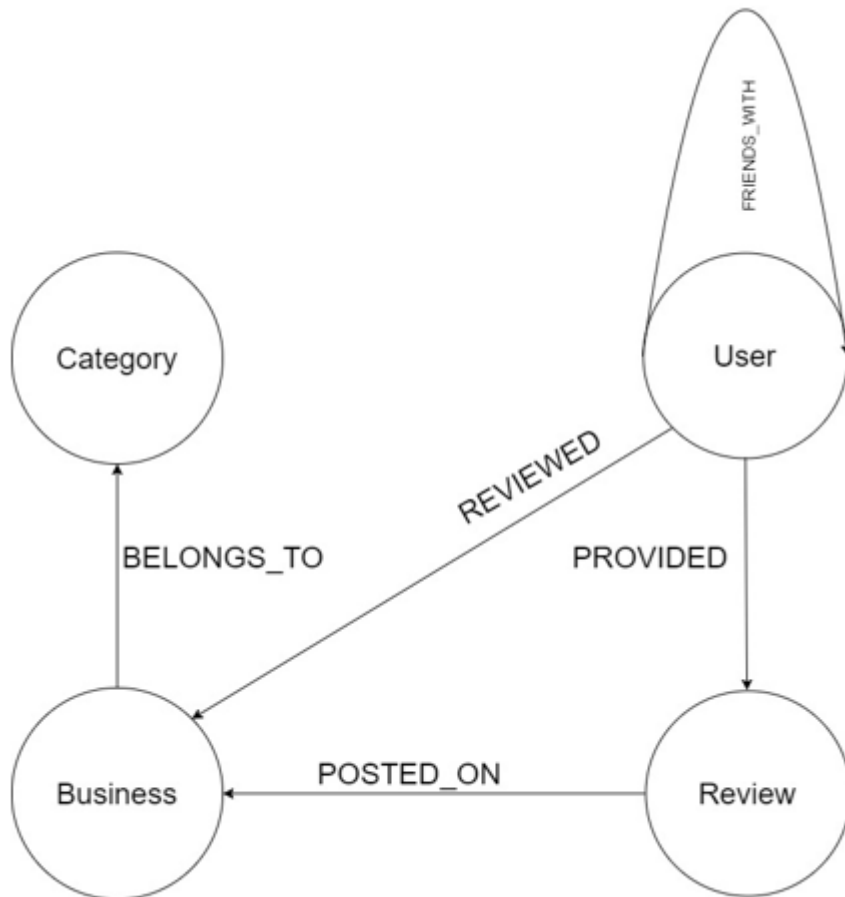
Document	Attribute	Data Format	MongoDB Data Type	Data Change	Constraint
Comment	Comment_ID	Text	String	Static	Unique, NotNull
	Text	Text	String	Static	Not Null
	Review_ID	Text	String	Static	Not Null
	User_ID	Text	String	Static	Not Null

Neo4j Pseudo Schema

The graph model prepared for the Amazon data set available to us is divided into 4 parts

- a) Nodes
- b) Properties of the Nodes
- c) Relationships
- d) Properties of the Relationships

Below is a simplified depiction of our graph model



Nodes

Nodes can be identified as entities that have a unique conceptual identity. In our graph model, we have included four types of nodes

- a) **Category(Unwinded from the Business collection)**
- b) **Business**
- c) **Review**
- d) **User**

Properties Of Category

The Category node will have the following properties: -

1. category_id - This property contains the unique identifier for the category.
2. category_title- This property contains the title of the category.

Properties Of Business

The Business node will have the following properties: -

1. business_id- This property contains the unique identifier for the business.
2. name- This property contains the name of the product.
3. neighborhood- This property contains the neighborhood to which the business belongs to.
4. city- This property contains the city to which the business belongs to.

5. state- This property contains the state to which the business belongs to.
6. postal_code- This property contains the postal code to which the business belongs to.
7. latitude- This property contains the latitude to which the business belongs to.
8. longitude- This property contains the longitude to which the business belongs to.
9. stars - This property contains the average rating of the business.
10. review_count - This property contains the total number of reviews posted on that business.
11. is_open- This property contains the Boolean value whether the business is currently open or not.
12. attributes- This property contains the various attributes of the business like PokestopNearby.
13. hours- This property contains the opening timings of the business

Properties Of Review

The Review node will have the following properties: -

1. review_id- The review will have a unique review_id that will be used as an identifier.
2. user_id – This property contains the unique identifier for the user who posted the review.
3. business_id – This property contains the unique identifier for the business for which the review has been posted.
4. stars – The rating awarded as part of this review.
5. date - This property contains the timestamp of the review in terms of day, month and year.
6. text – This property will contain the content or description of the review.
7. useful – This property will contain the number of users that voted this review as useful.
8. funny - This property will contain the number of users that voted this review as funny.
9. cool - This property will contain the number of users that voted this review as cool.
10. review_comments – This property will contain the comments that have been posted for the review.

Properties of User

The User node will have the following properties: -

1. user_id- The reviewer will have a unique user_id that will be used as an identifier.
2. name- The user will have it's own name as a separate property.
3. review_count- The total number of reviews posted by the user.
4. yelping_since- The date when the user signed up with Yelp.
5. friends- The friends of the user
6. useful- This property will contain the number of votes user has got as useful.
7. average_stars- The average rating provided by the user to different businesses.
8. badges- The badges awarded to the user.
9. location- The resident location of the user.

Relationships

Interactions between nodes are termed as relationships. In our graph model, we have included four types of relationships

- a) **“Belongs To” relationship between Business and Category.**
- b) **“Posted On” relationship between Review and Business.**
- c) **“Provided” relationship between User and Review.**
- d) **“Reviewed” relationship between User and Business.**
- e) **“Friends With” relationship between User and User.**

No Relationship will include any properties to be represented on our graph model

Data Preparation

- Selected two cities from USA - One from east coast and one from west coast.
- Cities chosen were North Las Vegas and Champaign.
- Dataset truncated to avoid memory issues.



We used the below queries to achieve this: -

1. `db.yelp_business.find({city:{$in:["North Las Vegas","Champaign"]}}).forEach(function(x){db.NLVCBusiness.insert(x)})`

This query collects the documents in the business collection that correspond to only the cities “North Las Vegas” and “Champaign”. It then creates a new collection called NLVCBusiness and inserts those documents in it.

```
> db.yelp_business.find({city:{$in:["North Las Vegas","Champaign"]}}).forEach( function(x){db.NLVCBusiness.insert(x)})
> db.NLVCBusiness.count()
2338
>
```

2. `db.yelp_review.find({business_id:{$in:db.NLVCBusiness.distinct("business_id")}}).forEach(function(x){db.NLVCReview.insert(x)})`

This query collects the documents in the review collection that correspond to only the distinct business_id present in the collection NLVCBusiness. It then creates a new collection called NLVCReview and inserts those documents in it.

```
> db.yelp_review.find({business_id:{$in:db.NLVCBusiness.distinct("business_id")}}).forEach( function(x){db.NLVCReview.in
sert(x)})
> db.NLVCReview.count()
57163
```

3. `db.yelp_user.find({user_id:{$in:db.NLVCReview.distinct("user_id")}}).forEach(function(x){db.NLVCUser.insert(x)})`

This query collects the documents in the user collection that correspond to only the distinct user_id present in the collection NLVCReview. It then creates a new collection called NLVCUser and inserts those documents in it.

```
> db.yelp_user.find({user_id:{$in:db.NLVCReview.distinct("user_id")}}).forEach( function(x){db.NLVUser.insert(x)})
> db.NLVUser.count()
28604
```

4. `db.yelp_checkin.find({business_id:{$in:db.NLVCBusiness.distinct("business_id")}}).forEach(function(x){db.NLVCheckin.insert(x)})`

This query collects the documents in the checkin collection that correspond to only the distinct `business_id` present in the collection `NLVCBusiness`. It then creates a new collection called `NLVCheckin` and inserts those documents in it.

```
> db.yelp_checkin.find({business_id:{$in:db.NLVCBusiness.distinct("business_id")}}).forEach( function(x){db.NLVCheckin.insert(x)})
> db.NLVCheckin.count()
1925
```

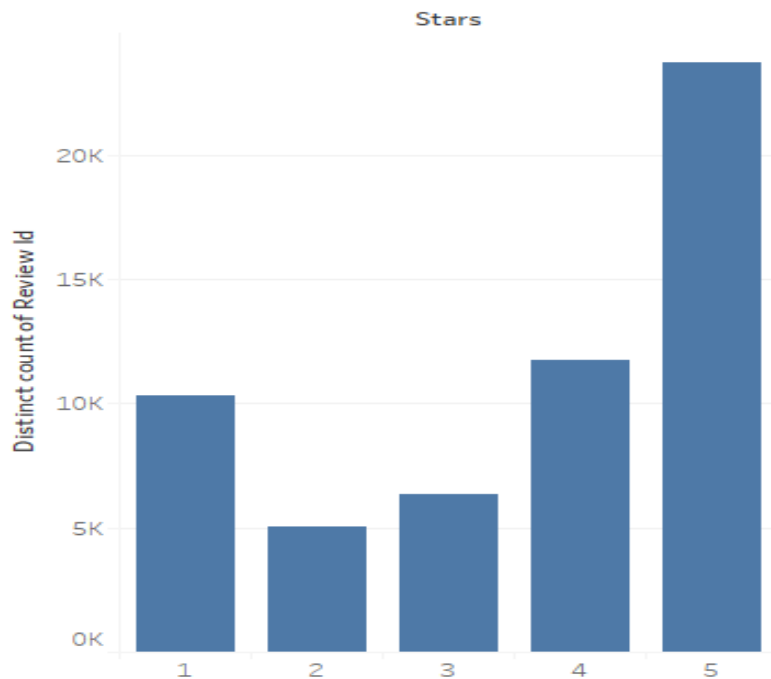
5. `db.yelp_tip.find({business_id:{$in:db.NLVCBusiness.distinct("business_id")}}).forEach(function(x){db.NLVCTip.insert(x)})`

This query collects the documents in the tip collection that correspond to only the distinct `business_id` present in the collection `NLVCBusiness`. It then creates a new collection called `NLVCTip` and inserts those documents in it.

```
> db.yelp_tip.find({business_id:{$in:db.NLVCBusiness.distinct("business_id")}}).forEach( function(x){db.NLVCTip.insert(x)})
> db.NLVCTip.count()
11801
```

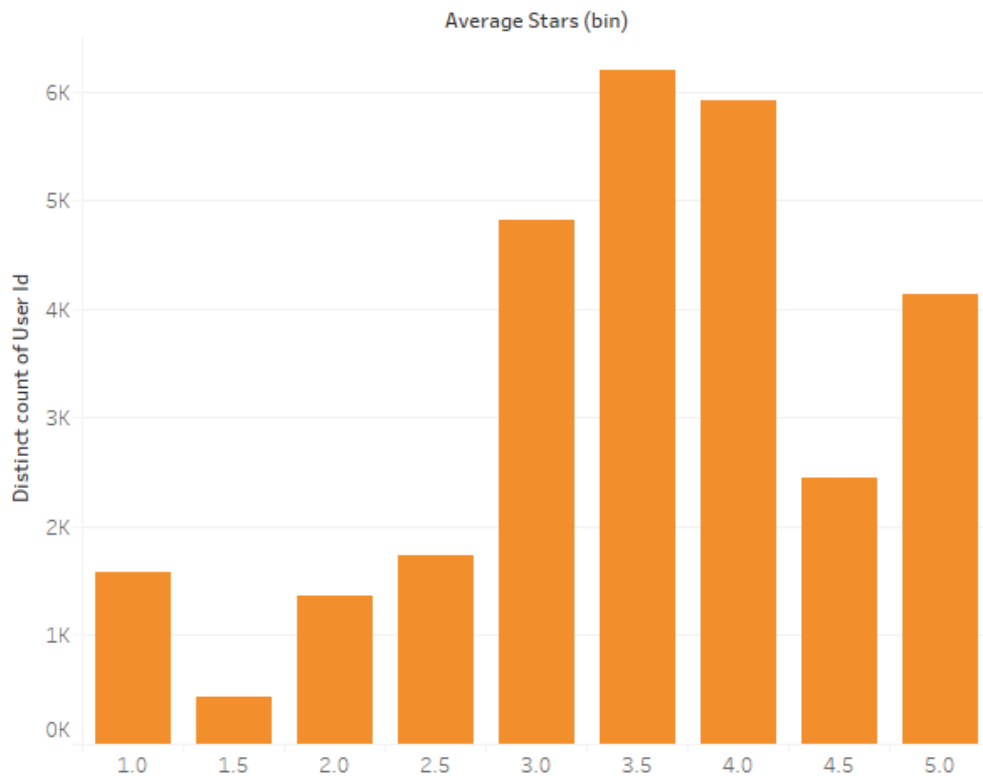

Overview of the Database through Visualizations

Rating Distribution of Review



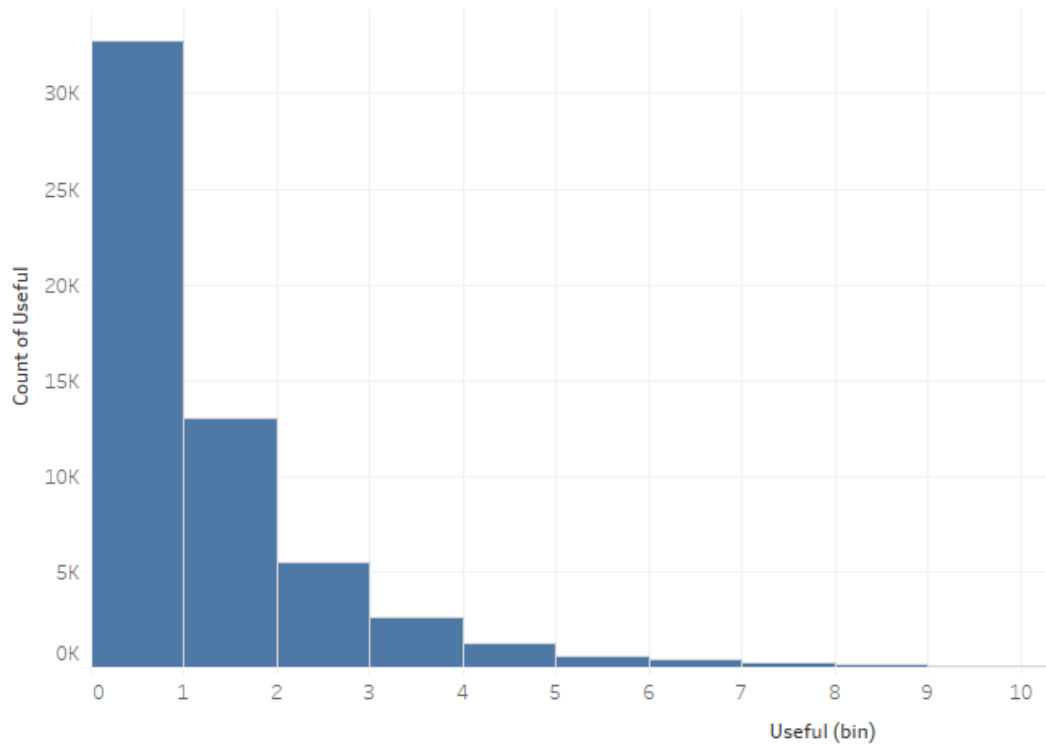
- The number of reviews corresponding to each review rating.
- Maximum number of reviews in our data set are rated 5 stars.

Rating Distribution of Users



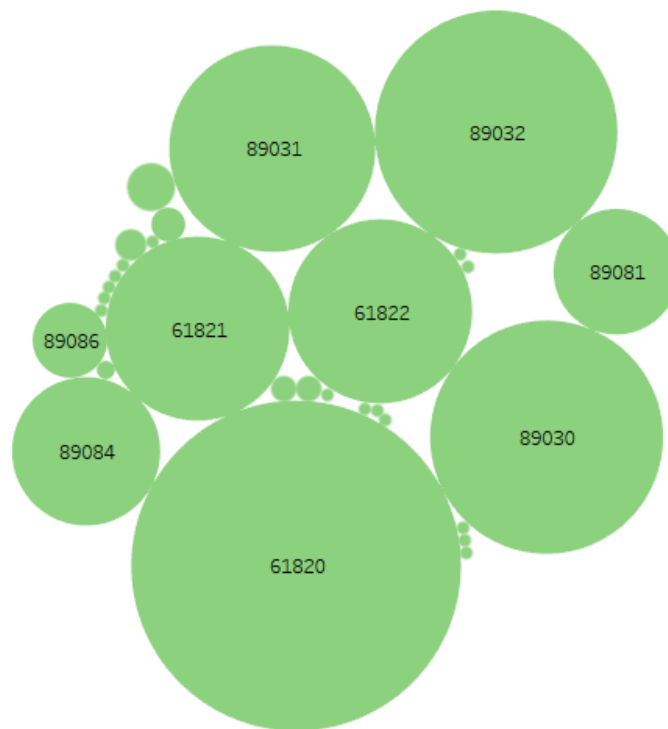
- The number of users corresponding to each user's average rating.
- Maximum number of users in our data set are rated approx. 3.5 stars.

Usefulness of Reviews



- The number of reviews corresponding to each bracket of usefulness.
- Maximum number of reviews fall in 0 and 1 bracket.

Postal Codes in North Las Vegas and Champaign



- Spread of number of reviews in the city of North Las Vegas.
- Maximum number of reviews fall in 61820.

Validation of Postal Codes with MongoDB

```
> db.NLVCBusiness.aggregate([{$group: {_id: {"pc": "$postal_code", "city": "$city"}, count: {$sum: 1}}}, {$sort: {count: -1}}])
{ "_id" : { "pc" : "61820", "city" : "Champaign" }, "count" : 663 }
{ "_id" : { "pc" : "89032", "city" : "North Las Vegas" }, "count" : 359 }
{ "_id" : { "pc" : "89030", "city" : "North Las Vegas" }, "count" : 331 }
{ "_id" : { "pc" : "89031", "city" : "North Las Vegas" }, "count" : 259 }
{ "_id" : { "pc" : "61822", "city" : "Champaign" }, "count" : 206 }
{ "_id" : { "pc" : "61821", "city" : "Champaign" }, "count" : 204 }
{ "_id" : { "pc" : "89084", "city" : "North Las Vegas" }, "count" : 134 }
{ "_id" : { "pc" : "89081", "city" : "North Las Vegas" }, "count" : 96 }
{ "_id" : { "pc" : "89086", "city" : "North Las Vegas" }, "count" : 34 }
{ "_id" : { "pc" : "89131", "city" : "North Las Vegas" }, "count" : 14 }
{ "_id" : { "pc" : "89130", "city" : "North Las Vegas" }, "count" : 7 }
{ "_id" : { "pc" : "89115", "city" : "North Las Vegas" }, "count" : 6 }
{ "_id" : { "pc" : "61801", "city" : "Champaign" }, "count" : 4 }
{ "_id" : { "pc" : "", "city" : "Champaign" }, "count" : 3 }
{ "_id" : { "pc" : "89085", "city" : "North Las Vegas" }, "count" : 2 }
{ "_id" : { "pc" : "61874", "city" : "Champaign" }, "count" : 1 }
{ "_id" : { "pc" : "89149", "city" : "North Las Vegas" }, "count" : 1 }
{ "_id" : { "pc" : "61825", "city" : "Champaign" }, "count" : 1 }
{ "_id" : { "pc" : "89132", "city" : "North Las Vegas" }, "count" : 1 }
{ "_id" : { "pc" : "89033", "city" : "North Las Vegas" }, "count" : 1 }
```

- Spread of number of reviews in the cities of North Las Vegas and Champaign.
- Maximum number of reviews fall in 61820.

Data Setup, Exploratory Analysis and Understanding the data

To make sure we don't run into memory issues while querying the Yelp database, we reduced our dataset to focus on only two cities i.e. North Las Vegas and Champaign.

We used the below queries to achieve this: -

Data setup Queries:

1. Query:

```
db.yelp_business.find({city:{$in:["North Las Vegas","Champaign"]}}).forEach(
function(x){db.NLVCBusiness.insert(x)})
```

Explanation:

This query collects the documents in the business collection that correspond to only the cities "North Las Vegas" and "Champaign". It then creates a new collection called NLVCBusiness and inserts those documents in it.

Screenshot:

```
> db.yelp_business.find({city:{$in:["North Las Vegas","Champaign"]}}).forEach( function(x){db.NLVCBusiness.insert(x)})
> db.NLVCBusiness.count()
2338
>
```

2. Query:

```
db.yelp_review.find({business_id:{$in:db.NLVCBusiness.distinct("business_id")}}).forEach(
function(x){db.NLVCReview.insert(x)})
```

Explanation:

This query collects the documents in the review collection that correspond to only the distinct business_id present in the collection NLVCBusiness. It then creates a new collection called NLVCReview and inserts those documents in it.

Screenshot:

```
> db.yelp_review.find({business_id:{$in:db.NLVCBusiness.distinct("business_id")}}).forEach( function(x){db.NLVCReview.in
sert(x)})
> db.NLVCReview.count()
57163
```

3. Query:

```
db.yelp_user.find({user_id:{$in:db.NLVCReview.distinct("user_id")}}).forEach(
function(x){db.NLVCUser.insert(x)})
```

Explanation:

This query collects the documents in the user collection that correspond to only the distinct user_id present in the collection NLVCReview. It then creates a new collection called NLVCUser and inserts those documents in it.

Screenshot:

```
> db.yelp_user.find({user_id:{$in:db.NLVCReview.distinct("user_id")}}).forEach( function(x){db.NLVCUser.insert(x)})
> db.NLVCUser.count()
28604
```

4. Query:

```
db.yelp_checkin.find({business_id:{$in:db.NLVCBusiness.distinct("business_id")}}).forEach(
function(x){db.NLVCCheckin.insert(x)})
```

Explanation:

This query collects the documents in the checkin collection that correspond to only the distinct business_id present in the collection NLVCBusiness. It then creates a new collection called NLVCCheckin and inserts those documents in it.

Screenshot:

```
> db.yelp_checkin.find({business_id:{$in:db.NLVCBusiness.distinct("business_id")}}).forEach( function(x){db.NLVCCheckin.
insert(x)})
> db.NLVCCheckin.count()
1925
```

5. Query:

```
db.yelp_tip.find({business_id:{$in:db.NLVCBusiness.distinct("business_id")}}).forEach(
function(x){db.NLVCTip.insert(x)})
```

Explanation:

This query collects the documents in the tip collection that correspond to only the distinct business_id present in the collection NLVCBusiness. It then creates a new collection called NLVCTip and inserts those documents in it.

Screenshot:

```
> db.yelp_tip.find({business_id:{$in:db.NLVCBusiness.distinct("business_id")}}).forEach( function(x){db.NLVCTip.insert(x)})
> db.NLVCTip.count()
11801
```

Basic understanding of the data/ exploratory Analysis:

Using these queries, we are trying to build a basic understanding of our data. Here, we are performing data exploration activity, to get a basic understanding of what the datatypes are, how big is our data, what kind of values are present for any document. How big is the size of one document, and how big is the collection.

1. Overall number of businesses**Query:**

```
db.NLVCBusiness.find().count()
```

```
2338
```

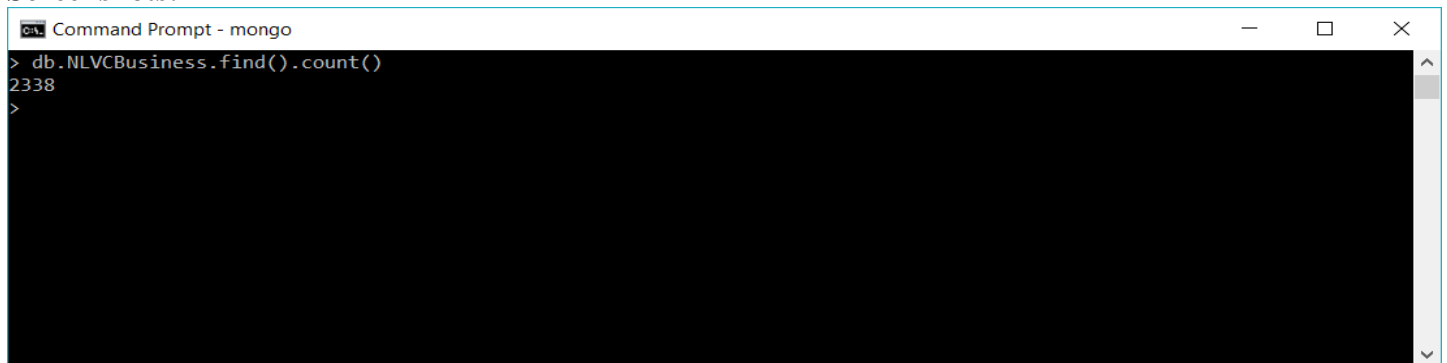
```
db.NLVCBusiness.aggregate([{$group: {_id: '$business_id', count: {$sum: 1}} }, {$group: {_id: null, count: {$sum:1}}}]])
```

```
{ "_id" : null, "count" : 2338 }
```

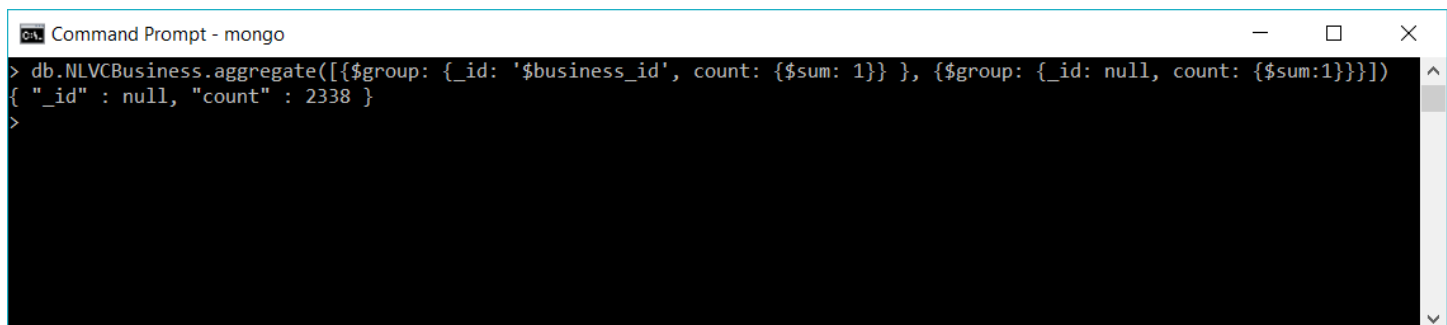
Explanation:

In this query we are trying to find the total number of “Business” in our collection. By “Business”, we are referring to a single isolated entity or organization, providing a product or a service, registered in the Yelp database.

This is done by counting the total number of unique rows in the collection “NLVCBusiness”, which can be done using the \$group stage in the aggregate pipeline using “business_id” as the identity. Although this number is same as the one we found out using a simple count, still the analysis was required to establish the fact that all the rows in the collection “NLVCBusiness” have a unique “business_id” field corresponding to one business.

Screenshots:

```
Command Prompt - mongo
> db.NLVCBusiness.find().count()
2338
>
```



```
Command Prompt - mongo
> db.NLVCBusiness.aggregate([{$group: {_id: '$business_id', count: {$sum: 1}} }, {$group: {_id: null, count: {$sum:1}}}]])
{ "_id" : null, "count" : 2338 }
>
```

2. Overall number of reviews

Query:

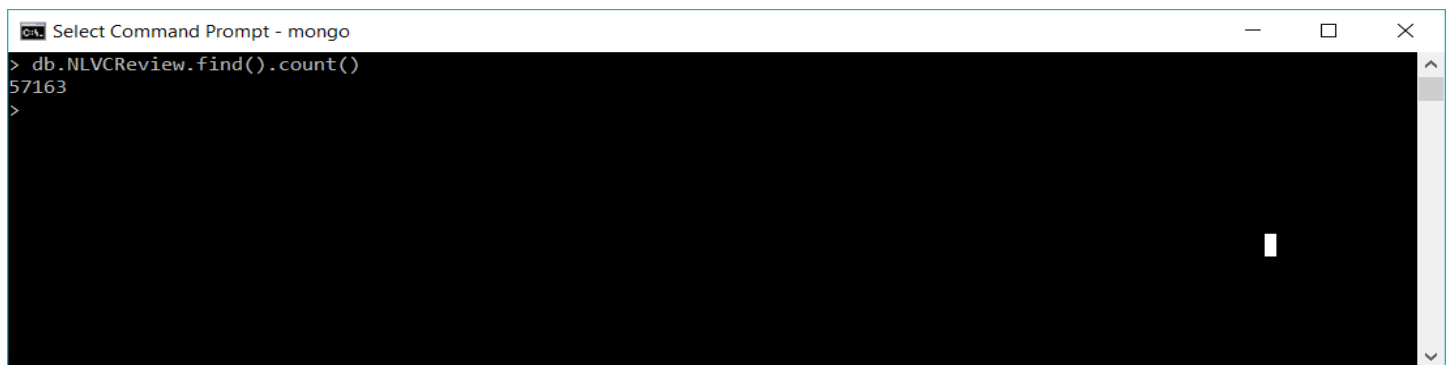
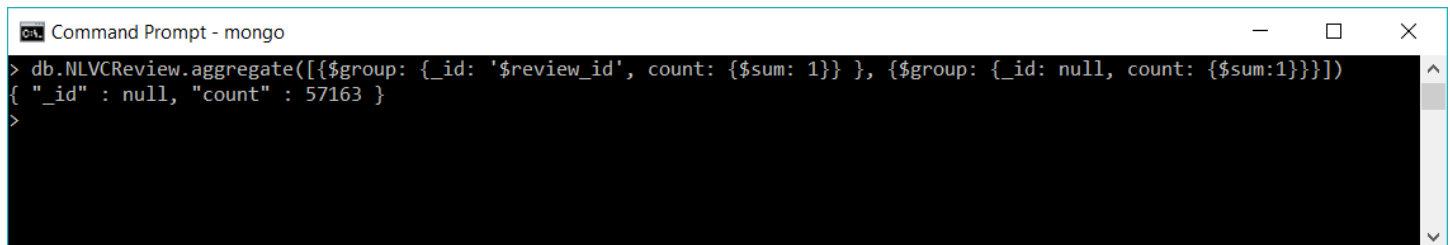
```
db.NLVCReview.find().count()
57163
```

```
db.NLVCReview.aggregate([{$group: {_id: '$review_id', count: {$sum: 1}} }, {$group: {_id: null, count: {$sum:1}}}]
{ "_id" : null, "count" : 57163 }
```

Explanation:

In this query we are finding the total number of reviews posted by a reviewer on a business present in our database. This is done by counting the total number of rows in the collection “NLVCReview”. Here we have followed a similar approach mentioned in (1) and used both the aggregate pipeline and the count command to find the number of reviews. Hence, we are finding the overall number of reviews with unique “review_id”.

Screenshot:

A screenshot of a MongoDB command prompt window titled "Select Command Prompt - mongo". The prompt shows the command `> db.NLVCReview.find().count()` being entered, followed by the output `57163` on the next line. The cursor is positioned at the end of the command line.A screenshot of a MongoDB command prompt window titled "Command Prompt - mongo". The prompt shows the command `> db.NLVCReview.aggregate([{$group: {_id: '$review_id', count: {$sum: 1}} }, {$group: {_id: null, count: {$sum:1}}}]` being entered, followed by the output `{ "_id" : null, "count" : 57163 }` on the next line. The cursor is positioned at the end of the command line.

3. Overall number of reviewers

Query:

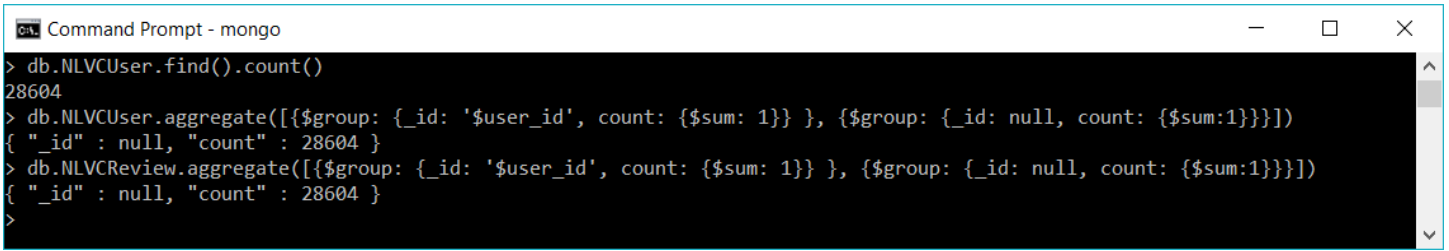
```
db.NLVCUser.find().count()
{ "_id" : null, "count" : 28604 }
```

```
db.NLVCReview.aggregate([{$group: {_id: '$user_id', count: {$sum: 1}} }, {$group: {_id: null, count: {$sum:1}}}]
{ "_id" : null, "count" : 28604 }
```

Explanation:

In this query we are trying to find the no of reviewers in our database using various metrics and comparing those values. So just like the above two queries, we have used a simple count approach. Secondly, we have used an approach of identifying the unique reviewers by using “\$group” stage on “userid”, and in the third approach, we have calculated, how many of these unique reviewers have posted atleast one review.

Screenshot:



```
Command Prompt - mongo
> db.NLVCUser.find().count()
28604
> db.NLVCUser.aggregate([{$group: {_id: '$user_id', count: {$sum: 1}} }, {$group: {_id: null, count: {$sum:1}}}]
{ "_id" : null, "count" : 28604 }
> db.NLVCReview.aggregate([{$group: {_id: '$user_id', count: {$sum: 1}} }, {$group: {_id: null, count: {$sum:1}}}]
{ "_id" : null, "count" : 28604 }
>
```

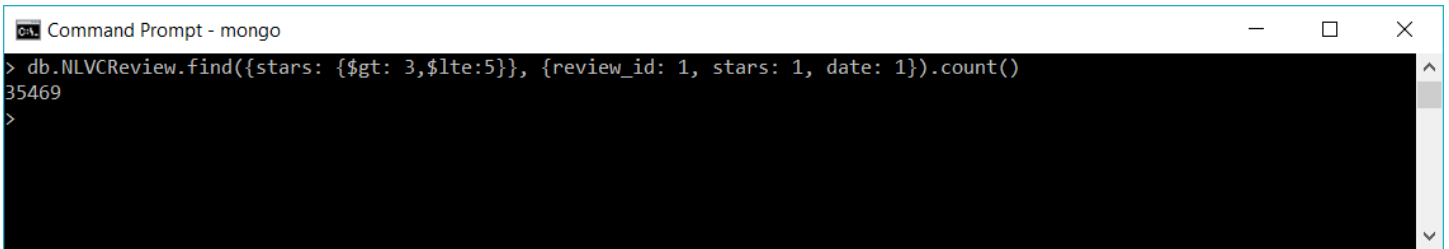
4. Overall number of reviews with ratings greater than 3

Query:

```
db.NLVCReview.find({stars: {$gt: 3,$lte:5}}, {review_id: 1, stars: 1, date: 1}).count()
35469
```

Explanation:

In this query we are trying to get an idea of how many positive reviews are present in our collection “NLVCReview”. We have gone with a rather simplistic approach here, because in this initial phase of exploratory analysis we are aiming to find in which direction the data is pointing, i.e. positive or negative. This is being achieved by using a simple count query along with a filter on the field “stars”. We have checked for the conditions greater than 3 to get the count of positive reviews and less than equals 5 to avoid capturing any garbage values.



```
Command Prompt - mongo
> db.NLVCReview.find({stars: {$gt: 3,$lte:5}}, {review_id: 1, stars: 1, date: 1}).count()
35469
>
```

5. Overall number of reviews with ratings less than 3

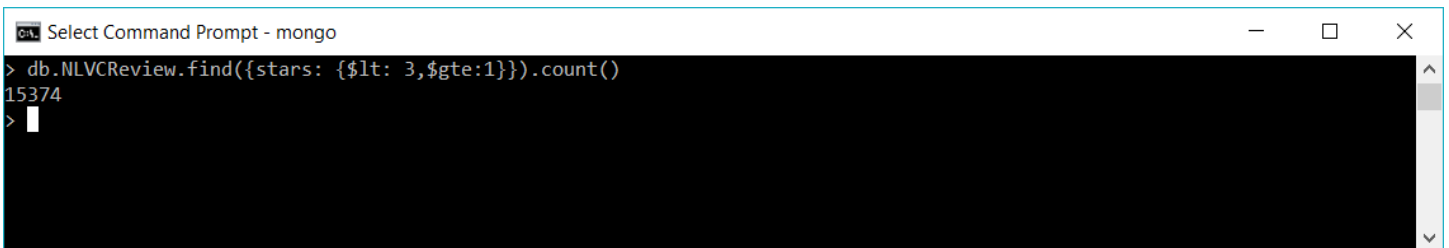
Query:

```
db.NLVCReview.find({stars: {$lt: 3,$gte:1}}).count()
15374
```

Explanation:

Just like the last query, in this query also we are trying to get an aggregate value of the negative sentiment amongst the reviewers, hence we have created a query to count the number of negative reviews, based on the ratings that have been provided. This we have achieved by simply writing a count query on top of a filter applied on the field “stars” having a value of only 1 and 2.

Screenshot:



```
Select Command Prompt - mongo
> db.NLVCReview.find({stars: {$lt: 3,$gte:1}}).count()
15374
>
```

6. Average number of reviews per business:

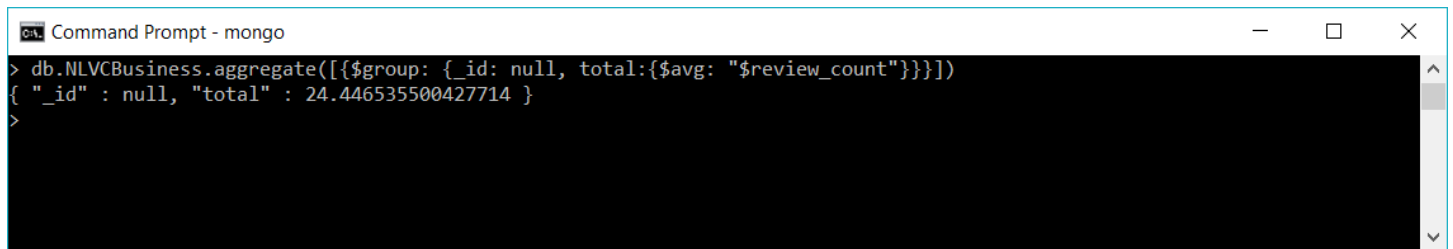
Query:

```
db.NLVCBusiness.aggregate([{$group: {_id: null, averageReviews: {$avg: "$review_count"}}}])
{ "_id" : null, "total" : 24.446535500427714 }
```

Explanation:

In this query, we are calculating the average number of reviews per business in our database, to get a general idea of how the reviews are being posted as a basic metric to judge if the businesses are getting enough reviews for analysis. In this query, we are using a group stage in the aggregate pipeline, with identity as null, to fetch all the values from the collection, and then subsequently taking the average of the review count.

Screenshot:



```
Command Prompt - mongo
> db.NLVCBusiness.aggregate([{$group: {_id: null, total: {$avg: "$review_count"}}}])
{ "_id" : null, "total" : 24.446535500427714 }
```

7. Total number of Categories

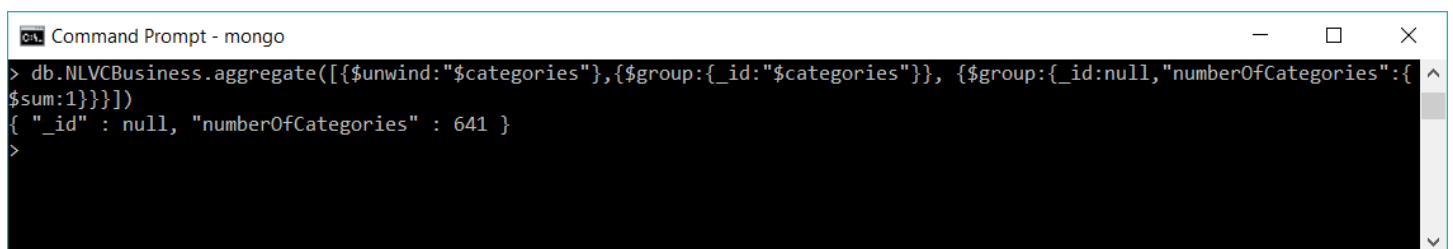
Query:

```
db.NLVCBusiness.aggregate([{$unwind: "$categories"}, {$group: {_id: "$categories"}}, {$group: {_id: null,
"numberOfCategories": { $sum: 1 }}}])
{ "_id" : null, "numberOfCategories" : 641 }
```

Categories:

By exploring the data, we found out that for each business is assigned to more than one category and all the categories have more than one businesses associated with them, hence there exists a many to many relation between businesses and categories. To explore this relation further we are querying to find the total number of categories that exists for all the businesses. In this query, first we are unwinding the array “categories”

Screenshot:



```
Command Prompt - mongo
> db.NLVCBusiness.aggregate([{$unwind: "$categories"}, {$group: {_id: "$categories"}}, {$group: {_id: null, "numberOfCategories": {
$sum: 1 }}}])
{ "_id" : null, "numberOfCategories" : 641 }
```

8. Date of the first review per category

Query:

```
db.NLVCReview.aggregate([{$group: {_id: "$business_id", date: {"$last": "$date"}}}, {$sort: {"date": 1}},
{$lookup: {from: "NLVCBusiness", localField: "_id", foreignField: "business_id", as: "Reference"}}, {$unwind:
"$Reference"}, {$unwind: "$Reference.categories"}, {$group: {_id: "$Reference.categories", date: {"$last":
"$date"}}}, {$sort: {"date": 1}}])
```

Explanation:

In this query, we are trying to extract information about the oldest review per category in our collection. From the above query, we already know that we have 641 categories in our database, we are interested in knowing the date when each of these categories received their first review.

Using this approach, we will be able to answer questions like, which categories are the oldest categories that exist, which categories are relatively new. Such metrics when combined with another query for the latest review and the number of reviews for that category, can give us idea about which Businesses are the oldest and still popular and trending, which businesses are not old, but currently trending or which business categories are not so much sought after.

Since the query was

1. In this query, first we created an aggregate pipeline on the collection “NLVCReview” and selected only the oldest review for each “business_id”, for this we used the sort function right after the group stage to get the oldest review for each “business_id”.
2. In the next step, we join the collection “NLVCReview” using a lookup on “business_id” with the collection “NLVCBusiness”, to get the category details for each business_id.
3. Once we have created a lookup, we need to unwind the created lookup first and then the category.
4. In the final stage, we are again using the group stage with identity field as “categories” combined with the sort function to get the oldest review date per category.

Screenshot:

```
Command Prompt - mongo
> db.NLVCReview.aggregate([{$group: {_id: "$business_id", date: {"$last": "$date"}}}, {$sort: {"date": 1}}, {$lookup: {from: "NLVCBusiness", localField: "_id", foreignField: "business_id", as: "Reference"}}, {$unwind: "$Reference"}, {$unwind: "$Reference.categories"}, {$group: {_id: "$Reference.categories", date: {"$last": "$date"}}}, {$sort: {"date": 1}}])
{ "_id" : "Dance Schools", "date" : "2007-01-08" }
{ "_id" : "Mini Golf", "date" : "2007-09-03" }
{ "_id" : "Trampoline Parks", "date" : "2007-09-03" }
{ "_id" : "Used Bookstore", "date" : "2009-07-17" }
{ "_id" : "Souvenir Shops", "date" : "2009-07-17" }
{ "_id" : "Personal Assistants", "date" : "2009-07-21" }
{ "_id" : "Vinyl Siding", "date" : "2009-08-24" }
{ "_id" : "Appraisal Services", "date" : "2009-10-01" }
{ "_id" : "Investing", "date" : "2009-10-25" }
{ "_id" : "Maternity Wear", "date" : "2010-04-11" }
{ "_id" : "Art Museums", "date" : "2010-04-25" }
{ "_id" : "Turkish", "date" : "2010-09-05" }
{ "_id" : "Gymnastics", "date" : "2010-12-31" }
{ "_id" : "Island Pub", "date" : "2011-01-29" }
{ "_id" : "Pakistani", "date" : "2011-04-07" }
{ "_id" : "Comic Books", "date" : "2011-04-08" }
{ "_id" : "Stadiums & Arenas", "date" : "2011-05-17" }
{ "_id" : "Seafood Markets", "date" : "2011-06-25" }
{ "_id" : "Buses", "date" : "2011-11-02" }
{ "_id" : "Fertility", "date" : "2012-05-16" }
Type "it" for more
>
```

9. The top 10 most prolific reviewers

Query:

```
db.NLVCReview.aggregate([{$group: {_id: {user_id: "$user_id"} , TotalReviews: {$sum: 1}}}, {$project: {user_id: 1, TotalReviews: 1}}, {$sort: {TotalReviews: -1}}, {$limit: 10}])
```

Explanation:

In this query, we are trying to fetch the most prolific reviewers which means the reviewers who are positing the most reviews. To extract the reviewers with the highest count of reviews, we have used the “\$group” stage with identity as “user_id” to get the sum of reviews for each reviewer. Subsequently, we have sorted this data in decreasing order of “TotalReviews”, and extracted the top 10 rows from this data. This will give us the top 10 most prolific reviewers.

Screenshot:

```
Command Prompt - mongo
> db.NLVCReview.aggregate([{$group: {_id: {user_id: "$user_id"} , TotalReviews: {$sum: 1}}}, {$project: {user_id: 1, TotalReviews: 1}}, {$sort: {TotalReviews: -1}}, {$limit: 10}])
{ "_id" : { "user_id" : "qntQp9UoeP6ju8D_W0Y0cw" }, "TotalReviews" : 141 }
{ "_id" : { "user_id" : "pFRE2mNCQvx9DUU542c6Dw" }, "TotalReviews" : 131 }
{ "_id" : { "user_id" : "qmjoMFMZdLH69_6eGTGDZw" }, "TotalReviews" : 111 }
{ "_id" : { "user_id" : "B8WruADju0mx5F1kLfYYfw" }, "TotalReviews" : 94 }
{ "_id" : { "user_id" : "C_ZQrG6VyPTtCwUHSdJu7w" }, "TotalReviews" : 90 }
{ "_id" : { "user_id" : "xN-xoLhTHUFTS_BMG00xg" }, "TotalReviews" : 84 }
{ "_id" : { "user_id" : "QZ_Arlwoj0ghfBvg69rj0w" }, "TotalReviews" : 81 }
{ "_id" : { "user_id" : "Fv0e9RIV9jw5TX3ctA1WbA" }, "TotalReviews" : 79 }
{ "_id" : { "user_id" : "y3FcL4bly0eLlkb0SDPnBQ" }, "TotalReviews" : 77 }
{ "_id" : { "user_id" : "GxkY7BqazkQu6I9HBDpbqQ" }, "TotalReviews" : 75 }
>
```

10. The top 10 most verbose reviewers

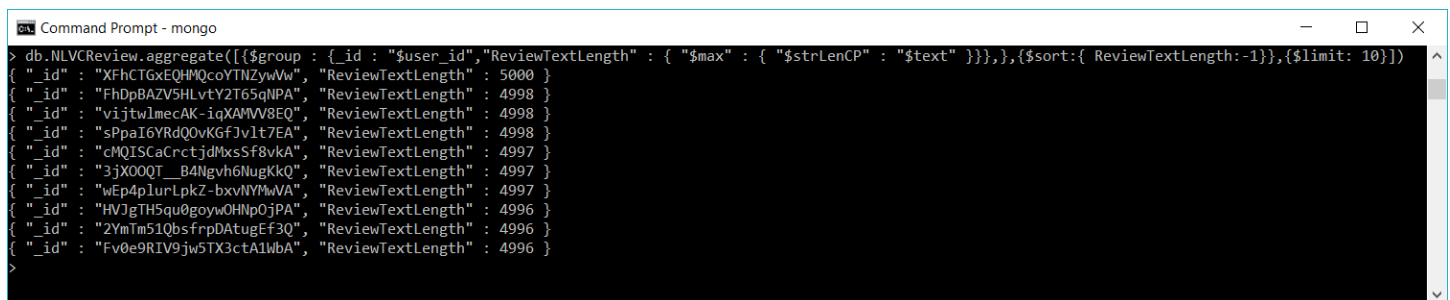
Query:

```
db.NLVCReview.aggregate([{$group : {_id : "$user_id","ReviewTextLength" : { "$max" : { "$strLenCP" : "$text" } } }},{$sort:{ ReviewTextLength:-1 }},{$limit: 10}])
```

Explanation:

In this query, we are trying to get the most verbose reviewers. Although this text length criteria for extracting the most verbose reviews is not so beneficial in itself, but during the prediction phase, such reviews become really helpful to create a pool of words for sentiment analysis of the reviews. Eg: we can select few of the most verbose reviews with rating 5 to train our model for positive sentiment and vice versa for negative sentiment.

In this query we are first using the “\$group” stage to group by “user_id” and in the same “\$group” stage, we are extracting the review with the maximum text length for that reviewer. Next we are sorting the data in decreasing order of text length to get the most verbose reviewers.



```
Command Prompt - mongo
> db.NLVCReview.aggregate([{$group : {_id : "$user_id","ReviewTextLength" : { "$max" : { "$strLenCP" : "$text" } } }},{$sort:{ ReviewTextLength:-1 }},{$limit: 10}])
{ "_id" : "XFhCTGxEQHMQcoYTINZywVw", "ReviewTextLength" : 5000 }
{ "_id" : "FhDpBAZV5HLvtY2T65qNPA", "ReviewTextLength" : 4998 }
{ "_id" : "viJtwImecAK-iqXAMV8EQ", "ReviewTextLength" : 4998 }
{ "_id" : "sPpaI6YRdQ0vKGFjvlt7EA", "ReviewTextLength" : 4998 }
{ "_id" : "cMQISCaCrctjdMxsSf8vkA", "ReviewTextLength" : 4997 }
{ "_id" : "3jX00QT_B4Ngvh6NugKkQ", "ReviewTextLength" : 4997 }
{ "_id" : "wEp4plurLpkZ-bxvNYMwVA", "ReviewTextLength" : 4997 }
{ "_id" : "HVJgTH5qu0goYwOHnpOjPA", "ReviewTextLength" : 4996 }
{ "_id" : "2YmTm51QbsfrpDATugEf3Q", "ReviewTextLength" : 4996 }
{ "_id" : "Fv0e9RIV9jw5TX3ctA1WbA", "ReviewTextLength" : 4996 }
```

11. Number of Businesses in each zip code:

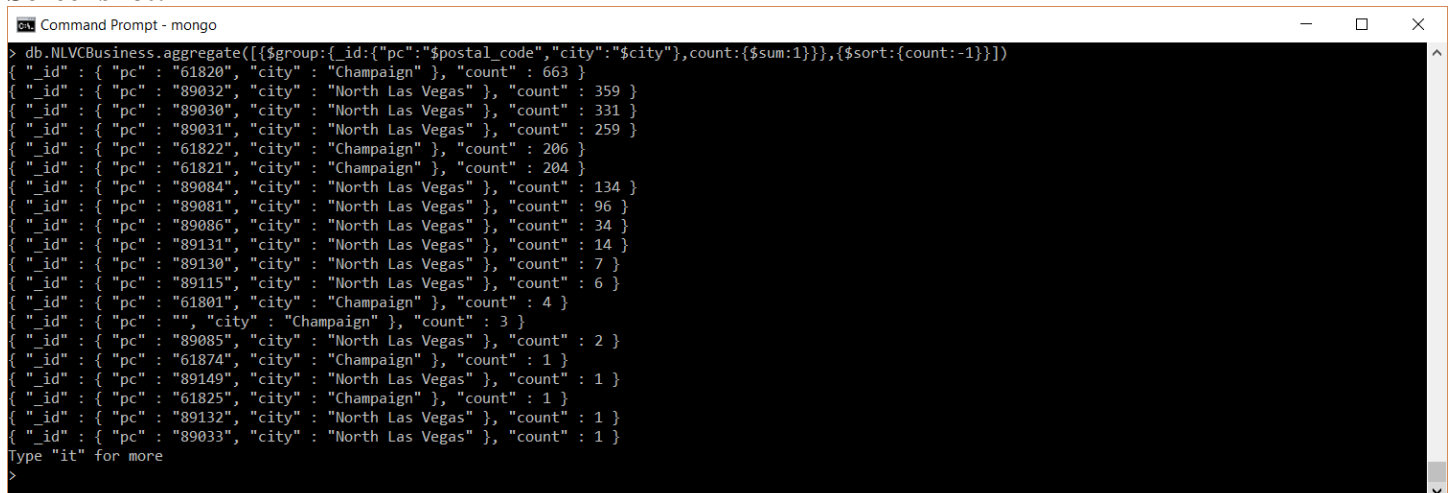
Query:

```
db.NLVCBusiness.aggregate([{$group: {_id: {"pc": "$postal_code", "city": "$city"}, count: { $sum: 1 } }},{$sort: { count:-1 } }])
```

Explanation:

This query aims to find out if most of the businesses are located in the same neighbourhood. Since this is the data exploration phase, we have used a simple grouping query, to count the number of businesses in each zip code. In this query, we are using the identity for our “\$group” stage as a combination of “postal_code” and “city” and sorting this data by the count of this combination.

Screenshot:



```
Command Prompt - mongo
> db.NLVCBusiness.aggregate([{$group: {_id: {"pc": "$postal_code", "city": "$city"}, count: { $sum: 1 } }},{$sort: { count:-1 } }])
{ "_id" : { "pc" : "61820", "city" : "Champaign" }, "count" : 663 }
{ "_id" : { "pc" : "89032", "city" : "North Las Vegas" }, "count" : 359 }
{ "_id" : { "pc" : "89030", "city" : "North Las Vegas" }, "count" : 331 }
{ "_id" : { "pc" : "89031", "city" : "North Las Vegas" }, "count" : 259 }
{ "_id" : { "pc" : "61822", "city" : "Champaign" }, "count" : 206 }
{ "_id" : { "pc" : "61821", "city" : "Champaign" }, "count" : 204 }
{ "_id" : { "pc" : "89084", "city" : "North Las Vegas" }, "count" : 134 }
{ "_id" : { "pc" : "89081", "city" : "North Las Vegas" }, "count" : 96 }
{ "_id" : { "pc" : "89086", "city" : "North Las Vegas" }, "count" : 34 }
{ "_id" : { "pc" : "89131", "city" : "North Las Vegas" }, "count" : 14 }
{ "_id" : { "pc" : "89130", "city" : "North Las Vegas" }, "count" : 7 }
{ "_id" : { "pc" : "89115", "city" : "North Las Vegas" }, "count" : 6 }
{ "_id" : { "pc" : "61801", "city" : "Champaign" }, "count" : 4 }
{ "_id" : { "pc" : "", "city" : "Champaign" }, "count" : 3 }
{ "_id" : { "pc" : "89085", "city" : "North Las Vegas" }, "count" : 2 }
{ "_id" : { "pc" : "61874", "city" : "Champaign" }, "count" : 1 }
{ "_id" : { "pc" : "89149", "city" : "North Las Vegas" }, "count" : 1 }
{ "_id" : { "pc" : "61825", "city" : "Champaign" }, "count" : 1 }
{ "_id" : { "pc" : "89132", "city" : "North Las Vegas" }, "count" : 1 }
{ "_id" : { "pc" : "89033", "city" : "North Las Vegas" }, "count" : 1 }
```

ANALYTICS

Yelp being a huge company in the B2B Service sector, is more concerned with categories than businesses. This is because of the fact that, an individual business performance would not bring any profit to Yelp as an organization. Analysing category trends and reviews will help it skew its business model towards the categories which are currently more profitable, so that they would be able to follow up with the social trends.

Analysis 1 – Popularity Trend Analysis

- a) The first metric that we have derived for analysis is a combination of 3 mongoDB queries. The first stage of this metric is calculation of the **first review in each category**. This will give us the date for the oldest review per category. We have already calculated such a query in the exploration phase.

Query:

```
db.NLVCReview.aggregate([
  {$group: {_id: "$business_id", date: {"$last": "$date"}}},
  {$sort: {"date": 1}},
  {$lookup: {from: "NLVCBusiness", localField: "_id", foreignField: "business_id", as: "Reference"}},
  {$unwind: "$Reference"}, {$unwind: "$Reference.categories"},
  {$group: {_id: "$Reference.categories", date: {"$last": "$date"}}},
  {$sort: {"date": 1}}])
```

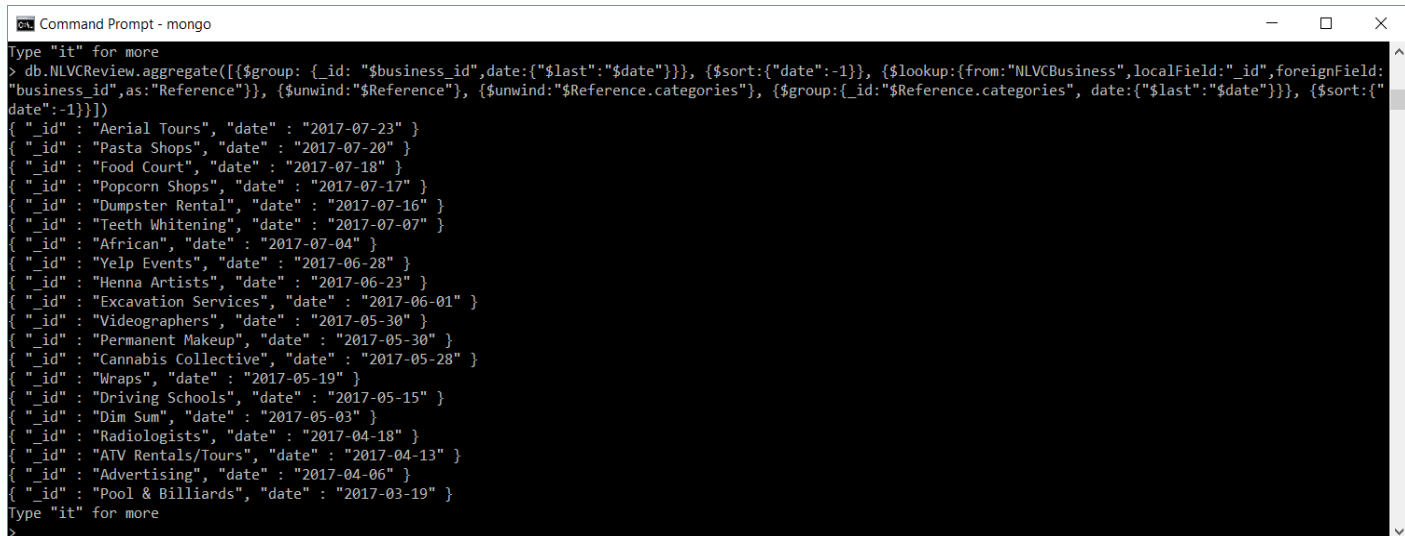
***Screenshot already attached with Data Exploration Queries - (Point 8)*

- b) **Extracting the date of the latest review per category.** For this query we have used the same approach followed in Query (8), except for the part where we have sorted this data in such a way that we will get the latest date. As discussed in the above query, using the queries – (8) and (9), one could derive a prediction, if the business is still trending or not and what is the oldest business category, which is still trending or what are the business categories that people are no longer talking about.

Query:

```
db.NLVCReview.aggregate([
  {$group: {_id: "$business_id", date: {"$last": "$date"}}},
  {$sort: {"date": -1}},
  {$lookup: {from: "NLVCBusiness", localField: "_id", foreignField: "business_id", as: "Reference"}},
  {$unwind: "$Reference"}, {$unwind: "$Reference.categories"},
  {$group: {_id: "$Reference.categories", date: {"$last": "$date"}}},
  {$sort: {"date": -1}}])
```

Screenshot:



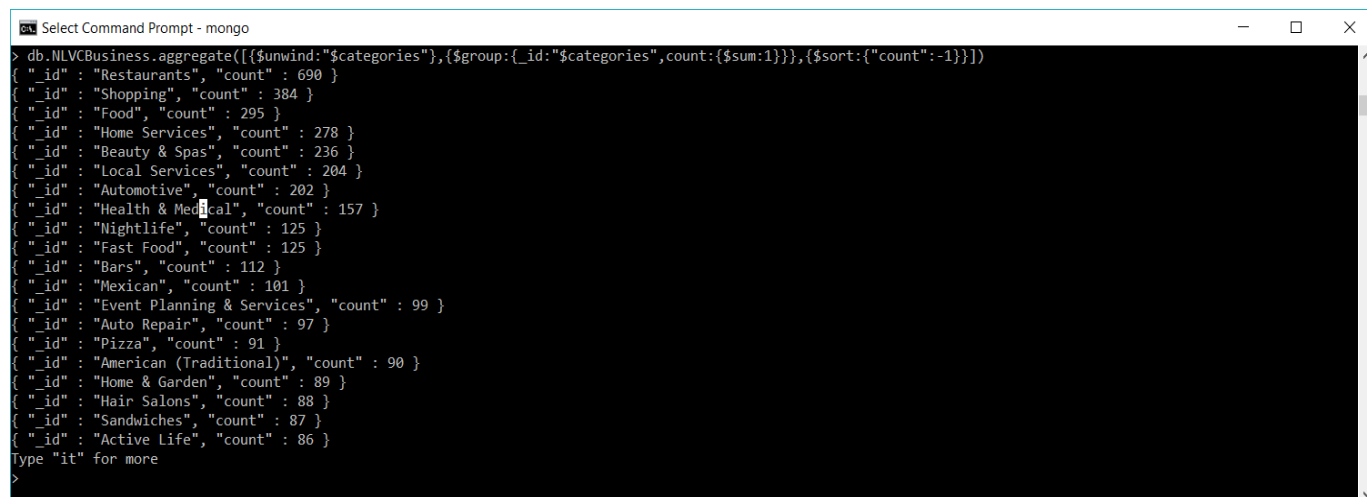
```
Command Prompt - mongo
Type "it" for more
> db.NLVCReview.aggregate([{$group: {_id: "$business_id", date: {"$last": "$date"}}}, {$sort: {"date": -1}}, {$lookup: {from: "NLVCBusiness", localField: "_id", foreignField: "business_id", as: "Reference"}}, {$unwind: "$Reference"}, {$unwind: "$Reference.categories"}, {$group: {_id: "$Reference.categories", date: {"$last": "$date"}}}, {$sort: {"date": -1}}])
{ "_id" : "Aerial Tours", "date" : "2017-07-23" }
{ "_id" : "Pasta Shops", "date" : "2017-07-20" }
{ "_id" : "Food Court", "date" : "2017-07-18" }
{ "_id" : "Popcorn Shops", "date" : "2017-07-17" }
{ "_id" : "Dumpster Rental", "date" : "2017-07-16" }
{ "_id" : "Teeth Whitening", "date" : "2017-07-07" }
{ "_id" : "African", "date" : "2017-07-04" }
{ "_id" : "Yelp Events", "date" : "2017-06-28" }
{ "_id" : "Henna Artists", "date" : "2017-06-23" }
{ "_id" : "Excavation Services", "date" : "2017-06-01" }
{ "_id" : "Videographers", "date" : "2017-05-30" }
{ "_id" : "Permanent Makeup", "date" : "2017-05-30" }
{ "_id" : "Cannabis Collective", "date" : "2017-05-28" }
{ "_id" : "Wraps", "date" : "2017-05-19" }
{ "_id" : "Driving Schools", "date" : "2017-05-15" }
{ "_id" : "Dim Sum", "date" : "2017-05-03" }
{ "_id" : "Radiologists", "date" : "2017-04-18" }
{ "_id" : "ATV Rentals/Tours", "date" : "2017-04-13" }
{ "_id" : "Advertising", "date" : "2017-04-06" }
{ "_id" : "Pool & Billiards", "date" : "2017-03-19" }
Type "it" for more
>
```

- c) **Calculating the number of reviews per category.** In this query we are trying to find the no of businesses in each category. We have already discussed about the utility of such an approach in query (8) and (9).

Here we are trying to predict which businesses were most talked about. When we combine this query with the results from query (8) and (9), we can make some powerful predictions like which businesses are trending and which businesses are not trending.

Query:

```
db.NLVCBusiness.aggregate([
  {$unwind:"$categories"},
  {$group: {_id:"$categories",count:{$sum:1}}},
  {$sort:{"count":-1}}])
```



```
> db.NLVCBusiness.aggregate([{$unwind:"$categories"},{$group: {_id:"$categories",count:{$sum:1}}},{$sort: {"count":-1}}])
{ "_id" : "Restaurants", "count" : 690 }
{ "_id" : "Shopping", "count" : 384 }
{ "_id" : "Food", "count" : 295 }
{ "_id" : "Home Services", "count" : 278 }
{ "_id" : "Beauty & Spas", "count" : 236 }
{ "_id" : "Local Services", "count" : 204 }
{ "_id" : "Automotive", "count" : 202 }
{ "_id" : "Health & Medical", "count" : 157 }
{ "_id" : "Nightlife", "count" : 125 }
{ "_id" : "Fast Food", "count" : 125 }
{ "_id" : "Bars", "count" : 112 }
{ "_id" : "Mexican", "count" : 101 }
{ "_id" : "Event Planning & Services", "count" : 99 }
{ "_id" : "Auto Repair", "count" : 97 }
{ "_id" : "Pizza", "count" : 91 }
{ "_id" : "American (Traditional)", "count" : 90 }
{ "_id" : "Home & Garden", "count" : 89 }
{ "_id" : "Hair Salons", "count" : 88 }
{ "_id" : "Sandwiches", "count" : 87 }
{ "_id" : "Active Life", "count" : 86 }
Type "it" for more
>
```

Summary - Analysis 1

According to our analysis, the combination of the above three queries, provide a powerful tool to predict which categories are trending. Using a combination of the above three queries, we aim to predict, which categories have been existing since a long time and still being talked about, which categories were once famous but are redundant now and which categories are one of the emerging categories. By providing such an analysis, we are directly predicting that which categories, Yelp as a business could invest in promoting.

Analysis 2: Geo-Spatial Queries

In this analysis, we are trying to predict, for any user, which restaurants near our user are currently open and these are sorted by the ratings of these businesses, to assist the user to make a better choice, most popular delivery options near my user or in our city.

- a) Creating a geo-spatial collection of document, by importing the location field with type as “Point” and the legacy coordinate pair (longitude, latitude)

Query A2.1

```
db.NLVCBusiness.updateMany({longitude:{$exists:true},latitude:{$exists:true}},{ $set : { location:
{type: "Point", coordinates: []}}})
```

Query A2.2

```
db.NLVCBusiness.find({longitude:{$exists:true},latitude:{$exists:true}}).forEach(
function(myDoc) {
myDoc.location.coordinates.push(myDoc.longitude);myDoc.location.coordinates.push(myDoc.latitu
de);db.NLVCBusiness.save(myDoc);} )
```

Screenshots:

Query A2.1

```
Command Prompt - mongo
> db.NLVCBusiness.updateMany({longitude:{$exists:true},latitude:{$exists:true}},{ $set : { location: {type: "Point", coordinates: []}}})
{ "acknowledged" : true, "matchedCount" : 2338, "modifiedCount" : 2338 }
>
```

```
Command Prompt - mongo
> db.NLVCBusiness.findOne()
{
  "_id" : ObjectId("5a15d5233fdc05ebb011841c"),
  "business_id" : "-2q4dnUw0gGJniGW2aPamQ",
  "name" : "Fiesta Ranchera",
  "neighborhood" : "",
  "address" : "1805 S Neil St",
  "city" : "Champaign",
  "state" : "IL",
  "postal_code" : "61820",
  "latitude" : 40.0940675,
  "longitude" : -88.2457853,
  "stars" : 2,
  "review_count" : 4,
  "is_open" : 0,
  "attributes" : {
    "GoodForMeal" : {
      "dessert" : false,
      "latenight" : false,
      "lunch" : false,
      "dinner" : false,
      "breakfast" : false,
      "brunch" : false
    },
    "RestaurantsGoodForGroups" : true,
    "NoiseLevel" : "quiet"
  },
  "categories" : [
    "Restaurants",
    "Mexican"
  ],
  "hours" : {
  },
  "location" : {
    "type" : "Point",
    "coordinates" : [ ]
  }
}
```

Query A2.2

```
Select Command Prompt - mongo
> db.NLVCBusiness.find({longitude:{$exists:true},latitude:{$exists:true}}).forEach( function(myDoc) { myDoc.location.coordinates.push(myDoc.longitude);myDoc.location.coordinates.push(myDoc.latitude);db.NLVCBusiness.save(myDoc);} )
>
```

```
Command Prompt - mongo
> db.NLVCBusiness.findOne()
{
  "_id" : ObjectId("5a15d5233fdc05ebb011841c"),
  "business_id" : "-2q4dnUw0gGJniGW2aPamQ",
  "name" : "Fiesta Ranchera",
  "neighborhood" : "",
  "address" : "1805 S Neil St",
  "city" : "Champaign",
  "state" : "IL",
  "postal_code" : "61820",
  "latitude" : 40.0940675,
  "longitude" : -88.2457853,
  "stars" : 2,
  "review_count" : 4,
  "is_open" : 0,
  "attributes" : {
    "GoodForMeal" : {
      "dessert" : false,
      "latenight" : false,
      "lunch" : false,
      "dinner" : false,
      "breakfast" : false,
      "brunch" : false
    },
    "RestaurantsGoodForGroups" : true,
    "NoiseLevel" : "quiet"
  },
  "categories" : [
    "Restaurants",
    "Mexican"
  ],
  "hours" : {
  },
  "location" : {
    "type" : "Point",
    "coordinates" : [
      -88.2457853,
      40.0940675
    ]
  }
}
```

b) Creating an index for running our Geo-Spatial Queries.

Query:

```
db.NLVCBusiness.createIndex({ location: "2dsphere" })
```

```
Select Command Prompt - mongo
> db.NLVCBusiness.createIndex({ location: "2dsphere" })
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
```

```
Command Prompt - mongo
> db.NLVCBusiness.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "yelpdb.NLVCBusiness"
  },
  {
    "v" : 2,
    "key" : {
      "location" : "2dsphere"
    },
    "name" : "location_2dsphere",
    "ns" : "yelpdb.NLVCBusiness",
    "2dsphereIndexVersion" : 3
  }
]
```

- c) **Restaurants currently open in a neighbourhood sorted by rating and review count.** This scenario is for identifying the order in which the businesses are going to be displayed to our user. Here we are using the geo-spatial query to identify the restaurants within a 5 mile radius from the user, and then we are sorting these businesses by their average stars and review count to get a list in decreasing order of stars, e.g. if a user searches for a restaurant, all the restaurants with the maximum rating and highest number of reviews, within a 5 mile radius will come first.

Query:

```
db.NLVCBusiness.find({ location:{ $nearSphere: { $geometry:{ type: "Point", coordinates: [ -88.2457853 , 40.0940675 ] },$maxDistance: 8046 } },is_open:{ $eq:1 } }).sort({ stars:-1,review_count:-1 }).limit(5).pretty()
```

```
Select Command Prompt - mongo
> db.NLVCBusiness.find({ location:{ $nearSphere: { $geometry:{ type: "Point", coordinates: [ -88.2457853 , 40.0940675 ] },$maxDistance: 8046 } },is_open:{ $eq:1 } }).sort({ stars:-1,review_count:-1 }).limit(5).pretty()
{
  "_id" : ObjectId("5a15d5533f3dc05eb012aab4"),
  "business_id" : "4hMDMvtfnpYV72_5QMbthA",
  "name" : "Old Time Meat & Deli Shoppe",
  "neighborhood" : "",
  "address" : "2018 S Neil St",
  "city" : "Champaign",
  "state" : "IL",
  "postal_code" : "61820",
  "latitude" : 40.091899,
  "longitude" : -88.2452807,
  "stars" : 5,
  "review_count" : 43,
  "is_open" : 1,
  "attributes" : {
    "RestaurantsTableService" : false,
    "GoodForMeal" : {
      "dessert" : false,
      "latenight" : false,
      "lunch" : false,
      "dinner" : false,
      "breakfast" : false,
      "brunch" : false
    },
    "Alcohol" : "none",
    "Caters" : false,
    "HasTV" : false,
    "RestaurantsGoodForGroups" : false,
    "NoiseLevel" : "average",
    "Wifi" : "no",
    "RestaurantsAttire" : "casual",
    "RestaurantsReservations" : false,
    "OutdoorSeating" : false,
    "BusinessAcceptsCreditCards" : true,
    "RestaurantsPriceRange2" : 2,
    "BikeParking" : true,
    "RestaurantsDelivery" : false,
  }
}
```

- d) **Restaurants currently open for delivery, in a 5 mile radius sorted sorted by rating and review count.** In this query we are trying to find those restaurants, which are still open and available for delivery within a 5-mile radius. Sometimes users are looking for a delivery option, while browsing on a website. Using this query we are providing the user this data, along with sorting based on ratings and review count to increase the reliability of our recommendation.

Query:

```
db.NLVCBusiness.find({ location:{ $nearSphere: { $geometry:{ type: "Point", coordinates: [ -88.2457853 , 40.0940675 ] } },$maxDistance: 8046 } }, "attributes.RestaurantsDelivery":true, is_open:{ $eq:1 } }).sort({ stars:-1,review_count:-1 }).limit(5).pretty()
```

```
Command Prompt - mongo
> db.NLVCBusiness.find({ location:{ $nearSphere: { $geometry:{ type: "Point", coordinates: [ -88.2457853 , 40.0940675 ] } },$maxDistance: 8046 } }, "attributes.RestaurantsDelivery":true,is_open:{ $eq:1 } }).sort({ stars:-1,review_count:-1 }).limit(5).pretty()
{
  "_id" : ObjectId("5a15d56a3fdc05ebb0131a46"),
  "business_id" : "KK0uK8bU5ksSuyCT6EUD5A",
  "name" : "A Hunt Design",
  "neighborhood" : "",
  "address" : "",
  "city" : "Champaign",
  "state" : "IL",
  "postal_code" : "61820",
  "latitude" : 40.1166102,
  "longitude" : -88.2412504,
  "stars" : 5,
  "review_count" : 15,
  "is_open" : 1,
  "attributes" : {
    "BusinessAcceptsCreditCards" : true,
    "RestaurantsPriceRange2" : 2,
    "BusinessAcceptsBitcoin" : false,
    "BikeParking" : true,
    "RestaurantsDelivery" : true,
    "WheelchairAccessible" : true
  },
  "categories" : [
    "Wedding Planning",
    "Party & Event Planning",
    "Shopping",
    "Flowers & Gifts",
    "Event Planning & Services",
    "Florists"
  ],
  "hours" : {
    "Monday" : "9:00-16:00",
    "Tuesday" : "9:00-16:00",
    "Friday" : "9:00-16:00",
    "Wednesday" : "9:00-16:00",
    "Thursday" : "9:00-16:00",
    "Saturday" : "10:00-16:00"
  },
  "location" : {
    "type" : "Point",
    "coordinates" : [
      -88.2412504,
      40.1166102
    ]
  }
}
```

Summary – Analysis 2

In this analysis, we have implemented the geo-spatial queries, so as to make the selection of restaurants more relevant to the user. Without the use of this query, it may occur that a user searches for a restaurant and the search result returns one from the other end of the country. Such results to be very annoying for the end user. Hence we are sorting the result by a location filter, before recommending a restaurant.

Analysis 3 – Reviewer Clustering Analysis -

This query is again a business centric query, where our business, might want to know which are most of the reviews are coming from or which who are the top 10 reviewers on their site. Here we are preparing various metrics to analyse the behaviour of reviewers in our system.

- In this query, we have decided not to take into account the metrics like : cool, funny and useful, because in our user database, the data is slightly spiked. Since we have trimmed down our data, to just 2 cities. These metrics like cool, useful and funny reflected the user's activity on the entire database. Hence for analysis for these 2 cities of North Las Vegas and Champaign, we chose to work on the Review Collection.

Query:

```
db.NLVCReview.aggregate([{$group: { _id:{user_id:"$user_id"} , TotalReviews: {$sum:1}}} ,{$project :{user_id: 1,TotalReviews:1 }},{ $sort:{TotalReviews:-1 }},{ $limit: 10}])
```

Screenshot:

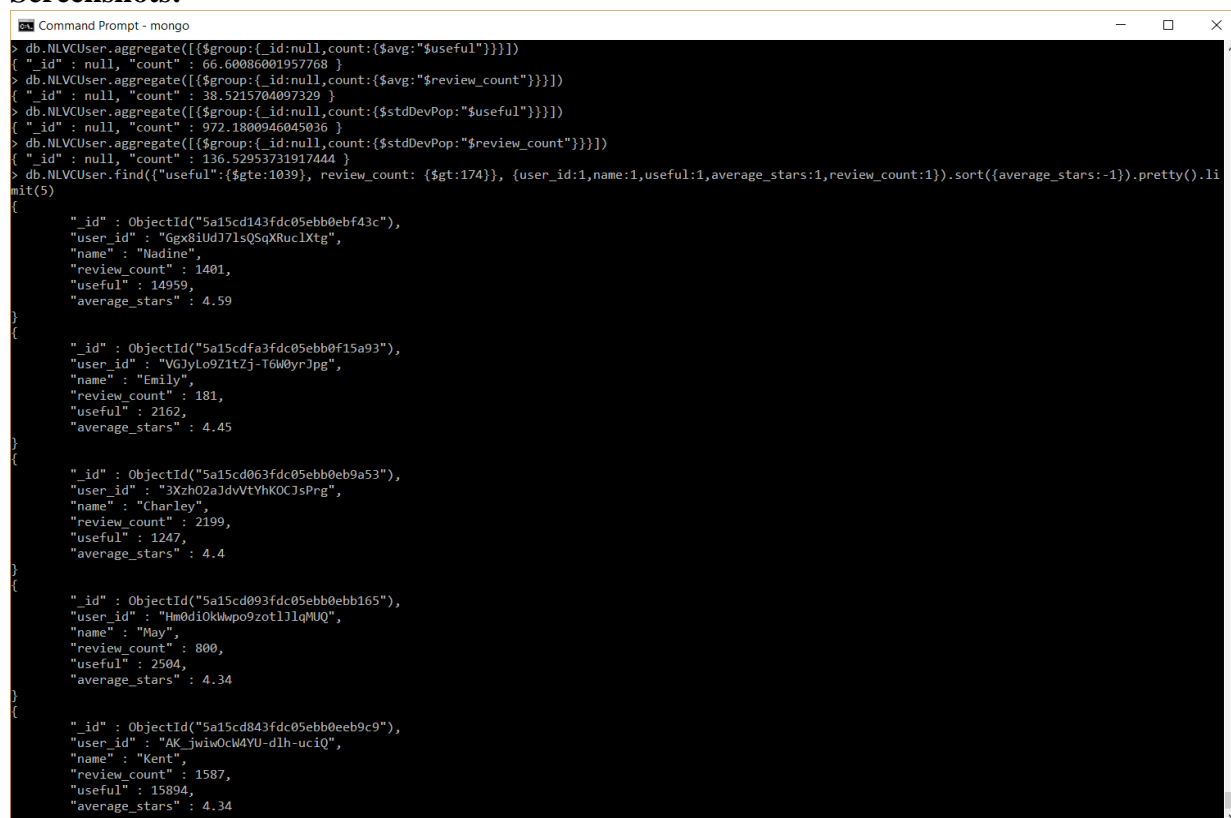
```
Command Prompt - mongo
> db.NLVCReview.aggregate([{$group: { _id:{user_id:"$user_id"} , TotalReviews: {$sum:1}}} ,{$project :{user_id: 1,TotalReviews:1 }},{ $sort:{TotalReviews:-1 }},{ $limit: 10}])
{ "_id" : { "user_id" : "qntQp9UoeP6ju8D_W0Y0cw" }, "TotalReviews" : 141 }
{ "_id" : { "user_id" : "pFRE2mNCQvx9DUU542c6Dw" }, "TotalReviews" : 131 }
{ "_id" : { "user_id" : "qmjoMFMZdLH69_6eGTGDZw" }, "TotalReviews" : 111 }
{ "_id" : { "user_id" : "B8WruADju0mx5F1kLfYYfw" }, "TotalReviews" : 94 }
{ "_id" : { "user_id" : "C_ZQrG6VyPTtCwUHSdJw7w" }, "TotalReviews" : 90 }
{ "_id" : { "user_id" : "xN-xoLhTHUFfTS_BMG60xg" }, "TotalReviews" : 84 }
{ "_id" : { "user_id" : "QZ_Arlwoj0ghf8Vg69rj0w" }, "TotalReviews" : 81 }
{ "_id" : { "user_id" : "Fv0e9RIV9jw5TX3ctA1WbA" }, "TotalReviews" : 79 }
{ "_id" : { "user_id" : "y3FcL4bly0eLlk0SDPnBQ" }, "TotalReviews" : 77 }
{ "_id" : { "user_id" : "GxkY7BqazkQu6I9H8DpbqQ" }, "TotalReviews" : 75 }
```


- b) In this query we are trying to extract the users with the highest average stars. Although the database contains a lot of users, which have a very high value of average stars, we want to filter out those reviewers, who have given sufficient reviews or who's reviews were not useful. Since usefulness is directly related to the quality of the review, it is safe to use "usefulness" and review_count as a metric for executing this query. This query will fetch us the most positive reviewers.

Query:

1. Mean of "useful" and "review_count":
`db.NLVCUser.aggregate([{$group:{_id:null,count:{$avg: "$useful"}}}])`
`db.NLVCUser.aggregate([{$group:{_id:null,count:{$avg: "$review_count"}}}])`
2. Standard deviation of "useful" and "review_count"
`db.NLVCUser.aggregate([{$group:{_id:null,count:{$stdDevPop: "$useful"}}}])`
`db.NLVCUser.aggregate([{$group:{_id:null,count:{$stdDevPop: "$review_count"}}}])`
3. We are assuming a normal distribution of review count and usefulness; hence we are filtering out the **top 16% of reviewers by taking a filter of mean + 1-standard deviation** on these fields.
`db.NLVCUser.find({"useful":{$gte:1039}, review_count: {$gt:174}},`
`{user_id:1,name:1,useful:1,average_stars:1,review_count:1}).`
`sort({average_stars:-1}).pretty().limit(5)`

Screenshots:



```
Command Prompt - mongo
> db.NLVCUser.aggregate([{$group:{_id:null,count:{$avg: "$useful"}}}])
{ "_id" : null, "count" : 66.60086001957768 }
> db.NLVCUser.aggregate([{$group:{_id:null,count:{$avg: "$review_count"}}}])
{ "_id" : null, "count" : 38.5215704097329 }
> db.NLVCUser.aggregate([{$group:{_id:null,count:{$stdDevPop: "$useful"}}}])
{ "_id" : null, "count" : 972.1800946045036 }
> db.NLVCUser.aggregate([{$group:{_id:null,count:{$stdDevPop: "$review_count"}}}])
{ "_id" : null, "count" : 136.52953731917444 }
> db.NLVCUser.find({"useful":{$gte:1039}, review_count: {$gt:174}}, {user_id:1,name:1,useful:1,average_stars:1,review_count:1}).sort({average_stars:-1}).pretty().limit(5)
{
  "_id" : ObjectId("5a15cd143fdc05ebb0ebf43c"),
  "user_id" : "Ggx81UdJ71sQsXruc1Xtg",
  "name" : "Nadine",
  "review_count" : 1481,
  "useful" : 14959,
  "average_stars" : 4.59
}
{
  "_id" : ObjectId("5a15cdfa3fdc05ebb0f15a93"),
  "user_id" : "VGjyLo9Z1tZj-T6W0yrJpg",
  "name" : "Emily",
  "review_count" : 181,
  "useful" : 2162,
  "average_stars" : 4.45
}
{
  "_id" : ObjectId("5a15cd063fdc05ebb0eb9a53"),
  "user_id" : "3XzhQ2aJdvYtYhKOC3sPrg",
  "name" : "Charley",
  "review_count" : 2199,
  "useful" : 1247,
  "average_stars" : 4.4
}
{
  "_id" : ObjectId("5a15cd093fdc05ebb0ebb165"),
  "user_id" : "Hm0di0k4Wpo9zotl1lqMUQ",
  "name" : "May",
  "review_count" : 800,
  "useful" : 2504,
  "average_stars" : 4.34
}
{
  "_id" : ObjectId("5a15cd843fdc05ebb0eeb9c9"),
  "user_id" : "AK_jwiwOCw4YU-dlh-uciQ",
  "name" : "Kent",
  "review_count" : 1587,
  "useful" : 15894,
  "average_stars" : 4.34
}
```

c) Clustering the oldest and most active reviewers

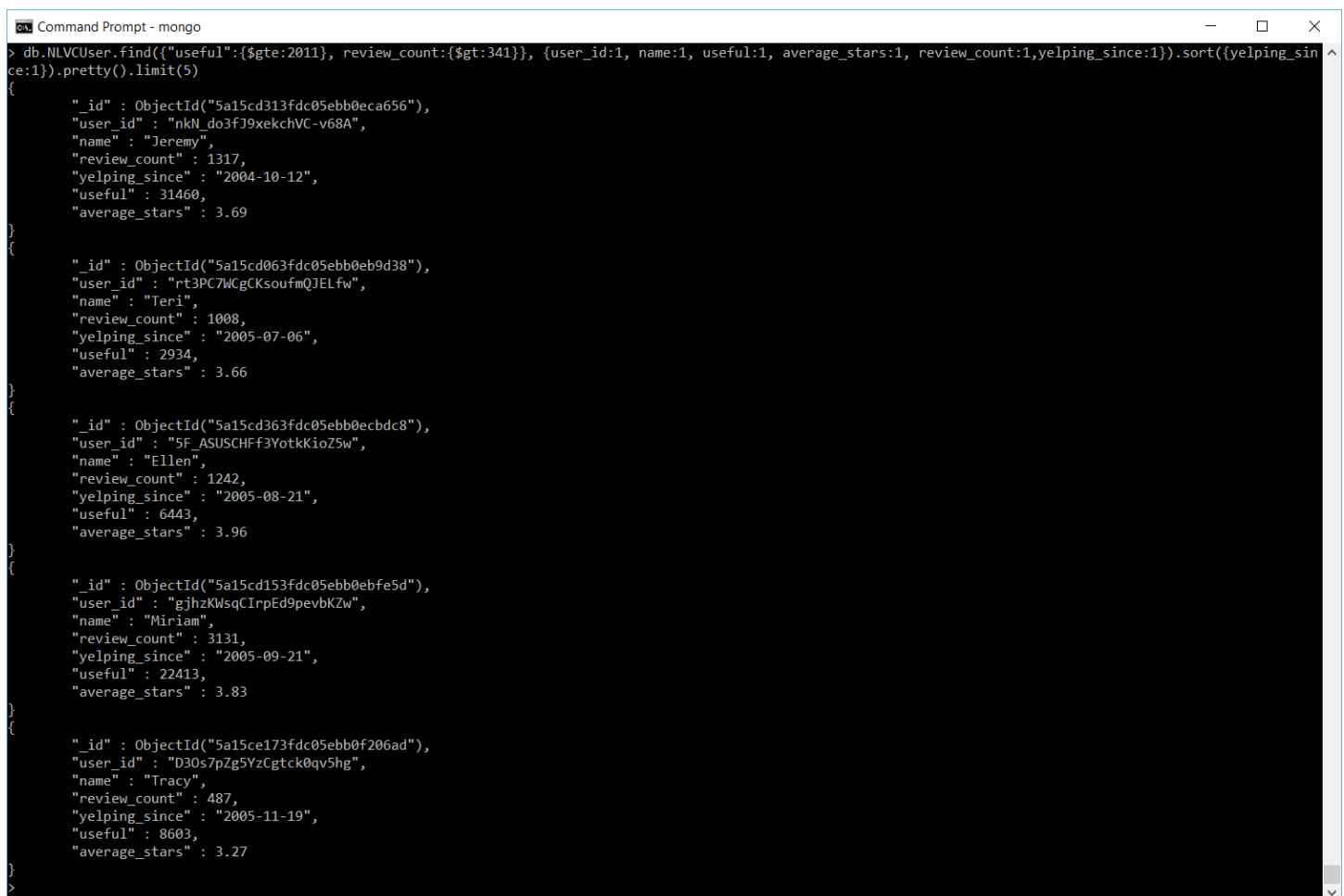
The analysis of those reviewers which have been reviewing for the longest time will help us get an idea of our review system. Further more their reviewing trends can be analysed, to give a review of the Yelp system. Often, without knowing, a user is reviewing the system too, along with the business, e.g. if amazon makes a mistake in their UI, a user might give low ratings to the affected product. Hence these reviewers, which have been reviewing for the longest time, will help us understand how our system has performed, by analysis of their average review rating and content over time.

Query:

```
db.NLVCUser.find({"useful":{$gte:2011}, review_count:{$gt:341}}, {user_id:1, name:1, useful:1, average_stars:1, review_count:1,yelping_since:1}).sort({yelping_since:1}).pretty().limit(10)
```


In this query we have assumed that, since the user is old, we have to only pick up those users, which have been in the ***top 2.5% people in terms of review count and usefulness***. We have achieved this by taking mean+2 standard deviation cut-off for the fields: “useful” and “review_count”

Screenshot:



```
Command Prompt - mongo
> db.NLVCUser.find({"useful":{"$gte:2011}}, review_count":{"$gt:341}}, {user_id:1, name:1, useful:1, average_stars:1, review_count:1,yelping_since:1}).sort({"yelping_since:1}).pretty().limit(5)
{
  "_id" : ObjectId("5a15cd313fdc05ebb0eca656"),
  "user_id" : "nkN_do3fJ9xekchVC-v68A",
  "name" : "Jeremy",
  "review_count" : 1317,
  "yelping_since" : "2004-10-12",
  "useful" : 31460,
  "average_stars" : 3.69
}
{
  "_id" : ObjectId("5a15cd063fdc05ebb0eb9d38"),
  "user_id" : "rt3PC7WCGCKsoufmQJELfw",
  "name" : "Teri",
  "review_count" : 1008,
  "yelping_since" : "2005-07-06",
  "useful" : 2934,
  "average_stars" : 3.66
}
{
  "_id" : ObjectId("5a15cd363fdc05ebb0ecbdc8"),
  "user_id" : "5F_ASUSCHff3YotkKioZ5w",
  "name" : "Ellen",
  "review_count" : 1242,
  "yelping_since" : "2005-08-21",
  "useful" : 6443,
  "average_stars" : 3.96
}
{
  "_id" : ObjectId("5a15cd153fdc05ebb0ebfe5d"),
  "user_id" : "gjhzKWsqCIrpEd9pevbKZw",
  "name" : "Miriam",
  "review_count" : 3131,
  "yelping_since" : "2005-09-21",
  "useful" : 22413,
  "average_stars" : 3.83
}
{
  "_id" : ObjectId("5a15ce173fdc05ebb0f206ad"),
  "user_id" : "D30s7pZg5YzCgtck0qv5hg",
  "name" : "Tracy",
  "review_count" : 487,
  "yelping_since" : "2005-11-19",
  "useful" : 8603,
  "average_stars" : 3.27
}
>
```

Analysis 4 -Text Based Helpfulness of a Review

Query:

```
db.NLVCReview.aggregate([{"$match": {$text: {$search : "visit recommend service parking customer delivery time -bitch", $diacriticSensitive: true}}}, {"$group": {_id:{$review_id:"$review_id"} }},{ $project: {review_id:1,text:1, score: { $meta: "textScore" } }},{ $sort: {score:-1 }},{ $limit: 10 }])
```

Explanation:

For this query we try to find the helpful reviews based on the text of each review. In this query, we are using an aggregate function where we are grouping based on the review_id. We have created a unique indexing for the document text. After creating the indexing, we have used the \$text method to search for helpful words like “visit recommend service parking customer delivery time” and exclude swear words like “bitch”. The \$text method by default ignores language-specific stop words, such as the and in English. We are also doing a diacritic sensitive search which includes accent marks as well. Based on the search for the helpful words, the \$meta function calculates the text score for the text field. In the end, we sort the reviews in the descending order of their scores.

```
> db.NLVCReview.aggregate([{"$match": {"$text": {"$search": "visit recommend service parking customer delivery time -bitch",
"$diacriticSensitive": true}}}, {"$group": {"_id": {"review_id": "$review_id"}}, {"$project": {"review_id": 1, "text": 1, "score": {"$meta":
"textScore"}}}, {"$sort": {"score": -1}}, {"$limit": 10}])
{"_id": {"review_id": "I38c0gQo96IQaFH279MDPw" }}
{"_id": {"review_id": "rx3n8KYbcvf5NiQe82nVPw" }}
{"_id": {"review_id": "60rePKjc308e-14Is_f0HA" }}
{"_id": {"review_id": "DfG118c01411WMk2Fyyitg" }}
{"_id": {"review_id": "wPpHzZxuY9f1diwC6vB-Lg" }}
{"_id": {"review_id": "PiimHh3A_96HmlEyxouUISA" }}
{"_id": {"review_id": "Im76muSvZReqJe0knp62YQ" }}
{"_id": {"review_id": "rT-sT8wPyos16r_NYzyH_w" }}
{"_id": {"review_id": "WvH-EWMJsb1TG_e0pK2wyw" }}
{"_id": {"review_id": "Y4wyA_Pr3-28Q5XHXsSdXTg" }}
```

Analysis 5 - Non-Text Based Helpfulness of a Review

Query:

```
db.NLVCReview.aggregate([{$sort: { business_id:1,date:-1 }},{$limit:50},{"$match": {stars:
{$gte:2,$lte:4},useful:{$gte:4}}},{$group:{"_id":{"business_id":"$business_id"},reviews:{$push:{review:{reviewText:"$text",rating:"$stars"}}}}},{$project:{review:{$slice:["$reviews",10]}}},{$allowDiskUse:true}).pretty()
```

Explanation:

For this query, we have used the overall rating, i.e. stars to be between 2 and 4. Second criteria is based on the review date. The latest reviews are only being taken. The review date is calculated by sorting date in descending order and business identifier, i.e. business_id in ascending order. We are also selecting the reviews that have been voted useful by more than 4 votes. At the end, we are displaying by using project function. We are displaying reviews (by just taking first 10 reviews – this is done by using slice functionality). AllowDiskUse equals True is just to provide some temporary disk space when sort or group function may take more than 100 MB of memory. Pretty is just to show print the data in a proper format which is easy to understand and look.

```
> db.NLVCReview.aggregate([{$sort: { business_id:1,date:-1}},{$limit:50},{"$match": {stars: {$gte:2,$lte:4},useful:{$gte:4}},{$group:{"_id":{"business_id":"$business_id"},reviews:{$push:{review:{reviewText:"$text",rating:"$stars"}}}}},{$project:{review:{$slice:["$reviews",10]}}},{$allowDiskUse:true}).pretty()
{
  "_id" : {
    "business_id" : "-2q4dnUw0gGJniGW2aPamQ"
  },
  "review" : [
    {
      "review" : {
        "reviewText" : "I used to really like El Toro which was in the same location. Name change brought about other changes as well that I am not loco about.\n\nThe service and decor are the same. It is a big restaurant and we were here on a Friday night of Moms' weekend and we got seated right away. Yay!\n\nThe chips and salsa were good and we ordered an additional appetizer. The cheese quesadilla was on tortilla folded over some cheese. It tasted like it had been heated in the microwave. No sour cream, no lettuce or tomato garnish, no guacamole on the side. Boo!\n\nWho serves an appetizer like this?\n\nMy daughter and I both had the Carne asada which was pretty much icky. Thin, dry and over cooked even though I had ordered it medium rare. This piece of meat zoomed past medium rare ten minutes before they took it off the grill. I thought if I could put it in the tortilla with some other goodies, it might be a bit more edible. It took forever to get anyone's attention to fill our salsa bowl. We were both bummed and had coffee and split a fried ice cream to cheer ourselves up. Didn't really do the trick. :( \n\nI won't come again, Champaign has other options that are better - now I understand how we got a table so quickly.",
        "rating" : 3
      }
    }
  ]
}
```

Conclusion:

The text-based approach and non-text based approach are not in agreement for our current dataset. This is because there are many cases where a reviewer would have used helpful words to mention about his experience of visiting a business but it didn't gain traction amongst the other users of Yelp and hence was not able to garner enough ratings or useful votes or maybe posted at an earlier date. In an ideal use-case, both the approaches should go hand-in-hand so as to validate one's findings with the other's. Also, to improve the helpfulness of a review we can add a comments collection to the dataset like we have done in our pseudo schema to provide another flavour towards knowing helpfulness for a review.

Analysis 6 – Category wise Clustering

In this analysis again, we have focusses on businesses and tried to pin point those businesses which are most trending or businesses which are most positive. We have done a category-wise clustering in this part and aggregated the review count for each category. Our basic aim with this query is to provide Yelp with the most rending categories so, to provide a more profitable business solution.

Queries:

1. Queries for count of categories with positive ratings

```
db.NLVCBusiness.aggregate([{$match:{stars:{$gt:3}}},{ $unwind: '$categories'},{$group:{_id:
"$categories", count:{$sum: "$review_count"}}},{ $sort:{_id:1}}])
```

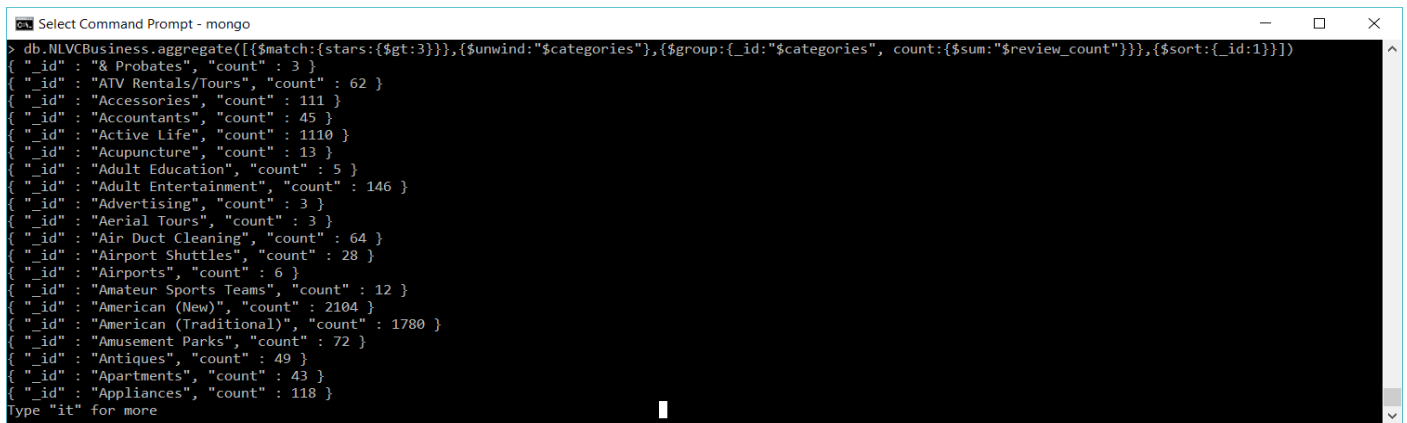
```
db.NLVCBusiness.aggregate([{$match:{stars:{$gt:3}}},{ $unwind: '$categories'},{$group:{_id:
"$categories", count:{$sum: "$review_count"}}},{ $sort:{count:1}}])
```

2. Queries for count of categories with negative ratings

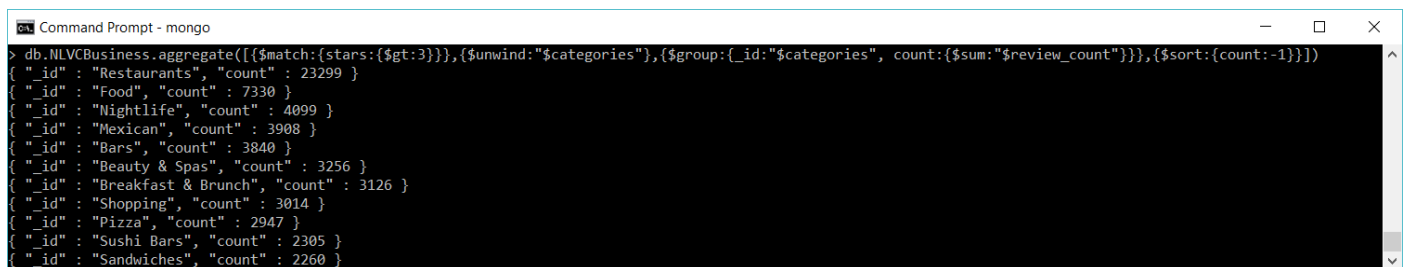
```
db.NLVCBusiness.aggregate([{$match:{stars:{$lt:3}}},{ $unwind: '$categories'},{$group:{_id:
"$categories", count:{$sum: "$review_count"}}},{ $sort:{_id:1}}])
```

```
db.NLVCBusiness.aggregate([{$match:{stars:{$lt:3}}},{ $unwind: '$categories'},{$group:{_id:
"$categories", count:{$sum: "$review_count"}}},{ $sort:{count:1}}])
```

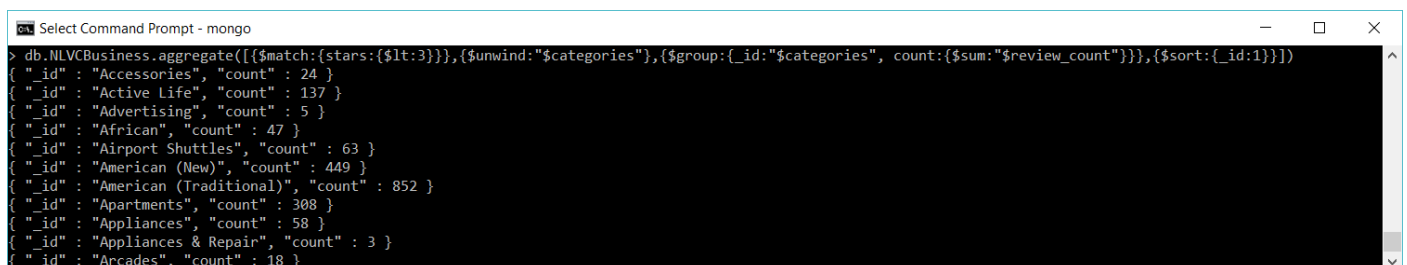
Screenshots:



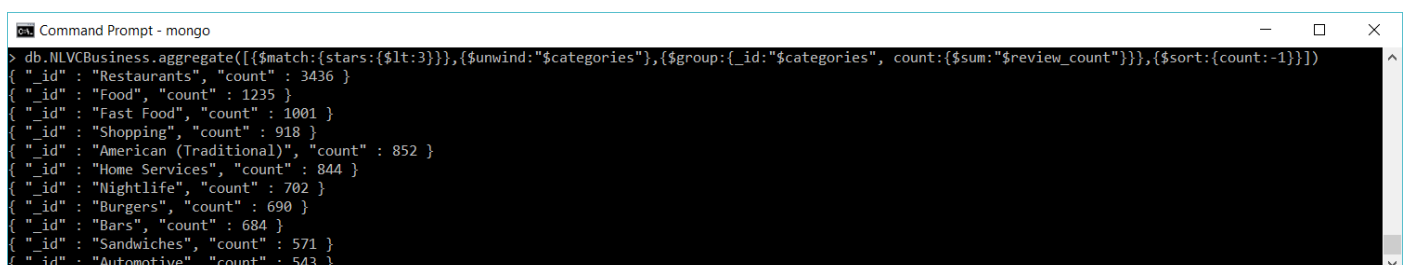
```
Select Command Prompt - mongo
> db.NLVCBusiness.aggregate([{$match:{stars:{$gt:3}}},{ $unwind: '$categories'},{$group:{_id: "$categories", count:{$sum: "$review_count"}}},{ $sort:{_id:1}}])
{ "_id" : "& Probates", "count" : 3 }
{ "_id" : "ATV Rentals/Tours", "count" : 62 }
{ "_id" : "Accessories", "count" : 111 }
{ "_id" : "Accountants", "count" : 45 }
{ "_id" : "Active Life", "count" : 1110 }
{ "_id" : "Acupuncture", "count" : 13 }
{ "_id" : "Adult Education", "count" : 5 }
{ "_id" : "Adult Entertainment", "count" : 146 }
{ "_id" : "Advertising", "count" : 3 }
{ "_id" : "Aerial Tours", "count" : 3 }
{ "_id" : "Air Duct Cleaning", "count" : 64 }
{ "_id" : "Airport Shuttles", "count" : 28 }
{ "_id" : "Airports", "count" : 6 }
{ "_id" : "Amateur Sports Teams", "count" : 12 }
{ "_id" : "American (New)", "count" : 2104 }
{ "_id" : "American (Traditional)", "count" : 1780 }
{ "_id" : "Amusement Parks", "count" : 72 }
{ "_id" : "Antiques", "count" : 49 }
{ "_id" : "Apartments", "count" : 43 }
{ "_id" : "Appliances", "count" : 118 }
Type "it" for more
```



```
Command Prompt - mongo
> db.NLVCBusiness.aggregate([{$match:{stars:{$gt:3}}},{ $unwind: '$categories'},{$group:{_id: "$categories", count:{$sum: "$review_count"}}},{ $sort:{count:-1}}])
{ "_id" : "Restaurants", "count" : 23299 }
{ "_id" : "Food", "count" : 7330 }
{ "_id" : "Nightlife", "count" : 4099 }
{ "_id" : "Mexican", "count" : 3908 }
{ "_id" : "Bars", "count" : 3840 }
{ "_id" : "Beauty & Spas", "count" : 3256 }
{ "_id" : "Breakfast & Brunch", "count" : 3126 }
{ "_id" : "Shopping", "count" : 3014 }
{ "_id" : "Pizza", "count" : 2947 }
{ "_id" : "Sushi Bars", "count" : 2305 }
{ "_id" : "Sandwiches", "count" : 2260 }
```



```
Select Command Prompt - mongo
> db.NLVCBusiness.aggregate([{$match:{stars:{$lt:3}}},{ $unwind: '$categories'},{$group:{_id: "$categories", count:{$sum: "$review_count"}}},{ $sort:{_id:1}}])
{ "_id" : "Accessories", "count" : 24 }
{ "_id" : "Active Life", "count" : 137 }
{ "_id" : "Advertising", "count" : 5 }
{ "_id" : "African", "count" : 47 }
{ "_id" : "Airport Shuttles", "count" : 63 }
{ "_id" : "American (New)", "count" : 449 }
{ "_id" : "American (Traditional)", "count" : 852 }
{ "_id" : "Apartments", "count" : 308 }
{ "_id" : "Appliances", "count" : 58 }
{ "_id" : "Appliances & Repair", "count" : 3 }
{ "_id" : "Arcades", "count" : 18 }
```



```
Command Prompt - mongo
> db.NLVCBusiness.aggregate([{$match:{stars:{$lt:3}}},{ $unwind: '$categories'},{$group:{_id: "$categories", count:{$sum: "$review_count"}}},{ $sort:{count:-1}}])
{ "_id" : "Restaurants", "count" : 3436 }
{ "_id" : "Food", "count" : 1235 }
{ "_id" : "Fast Food", "count" : 1001 }
{ "_id" : "Shopping", "count" : 918 }
{ "_id" : "American (Traditional)", "count" : 852 }
{ "_id" : "Home Services", "count" : 844 }
{ "_id" : "Nightlife", "count" : 702 }
{ "_id" : "Burgers", "count" : 690 }
{ "_id" : "Bars", "count" : 684 }
{ "_id" : "Sandwiches", "count" : 571 }
{ "_id" : "Automotive", "count" : 543 }
```

Neo4j Data Analytics

Data Import:

We used APOC plug in to import our datasets. We have used Apoc.load.json to import our json files. The queries to import our NeoBusiness.json, NeoUser.json, NeoReview.json are shown below in the same order.

```
CALL apoc.load.json("file:/Users/Admin/Desktop/MSIS/Unstructured%20Data/yelp_two_cities_dataset/NeoBusiness.json") YIELD value AS business MERGE(b:Business{business_id:business.business_id})
SET b.address = business.address,
b.latitude = business.latitude,
b.longitude = business.longitude,
b.name = business.name,
b.city = business.city,
b.postal_code = business.postal_code,
b.state = business.state,
b.review_count = business.review_count,
b.stars = business.stars,
b.neighborhood = business.neighborhood
WITH b, business.categories AS categories
UNWIND categories as cat
MERGE (c:Category {name: cat})
MERGE (b)-[:BELONGS_TO]->(c)
```

```
CALL apoc.load.json("file:/Users/Admin/Desktop/MSIS/Unstructured%20Data/yelp_two_cities_dataset/NeoUser.json") YIELD value AS user
MERGE(u:User{user_id:user.user_id})
SET u.name = user.name,
u.review_count = user.review_count,
u.yelping_since = user.yelping_since,
u.useful = user.useful,
u.fans = user.fans,
u.average_stars = user.average_stars
```

```
CALL apoc.load.json("file:/Users/Admin/Desktop/MSIS/Unstructured%20Data/yelp_two_cities_dataset/NeoReview.json") YIELD value AS review
MERGE(r:Review{review_id:review.review_id})
SET r.stars = review.stars,
r.user_id = review.user_id,
r.business_id = review.business_id,
r.date = review.date,
r.text = review.text,
r.useful = review.useful
```

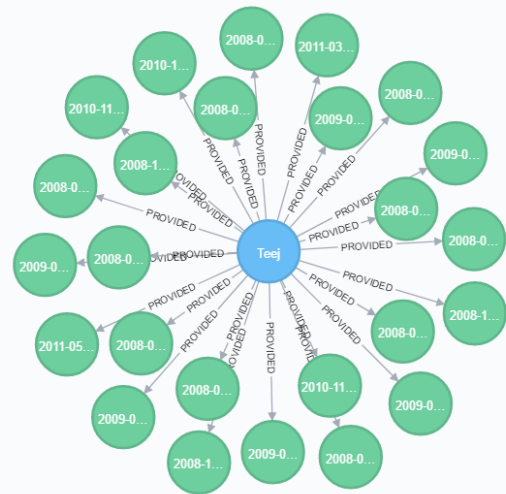
Relationships:

1. The first relationship is created between user and Review. This relationship is termed as 'Provided'. The query to create this relationship is shown below along with its output.

```
MATCH (u:User), (r:Review)
WHERE u.user_id=r.user_id
CREATE (u)-[:PROVIDED]->(r)
```

*(26) User(1) Review(25)

*(25) PROVIDED(25)

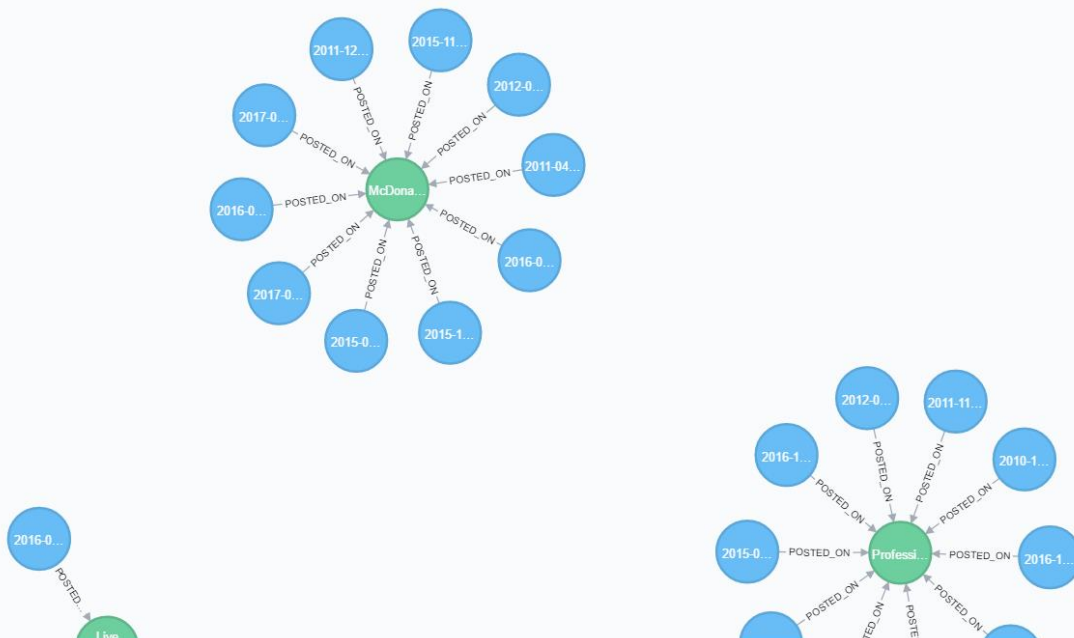


- The second relationship is created as 'Posted On' between Business and Review. Its query and output is shown below.

```
MATCH (b:Business),(r:Review)
WHERE b.business_id=r.business_id
CREATE (r)-[:POSTED_ON]->(b)
```

*(29) Review(25) Business(4)

*(25) POSTED_ON(25)



- The third relationship is what we created to optimize our research and find a direct relationship between Business and user. We created a 'Reviewed' relation between Business and User. Its query and output are shown below.

```
MATCH (b:Business),(c:User),(d:Review)
WHERE b.business_id=d.business_id and c.user_id = d.user_id
```

CREATE (b)-[:Reviewed]->(c)

(26) User(1) Business(25)

(25) REVIEWED(25)



4. Fourth and the last relationship that we created is between Business and Category. Its termed as “Belongs to”. The query and output is shown below.

```
MATCH (b:Business),(c:Category)
WHERE b.business_id=c.business_id
CREATE (r)-[:Belongs_to]->(c)
```

(26) Business(25) Category(1)

(25) BELONGS_TO(25)

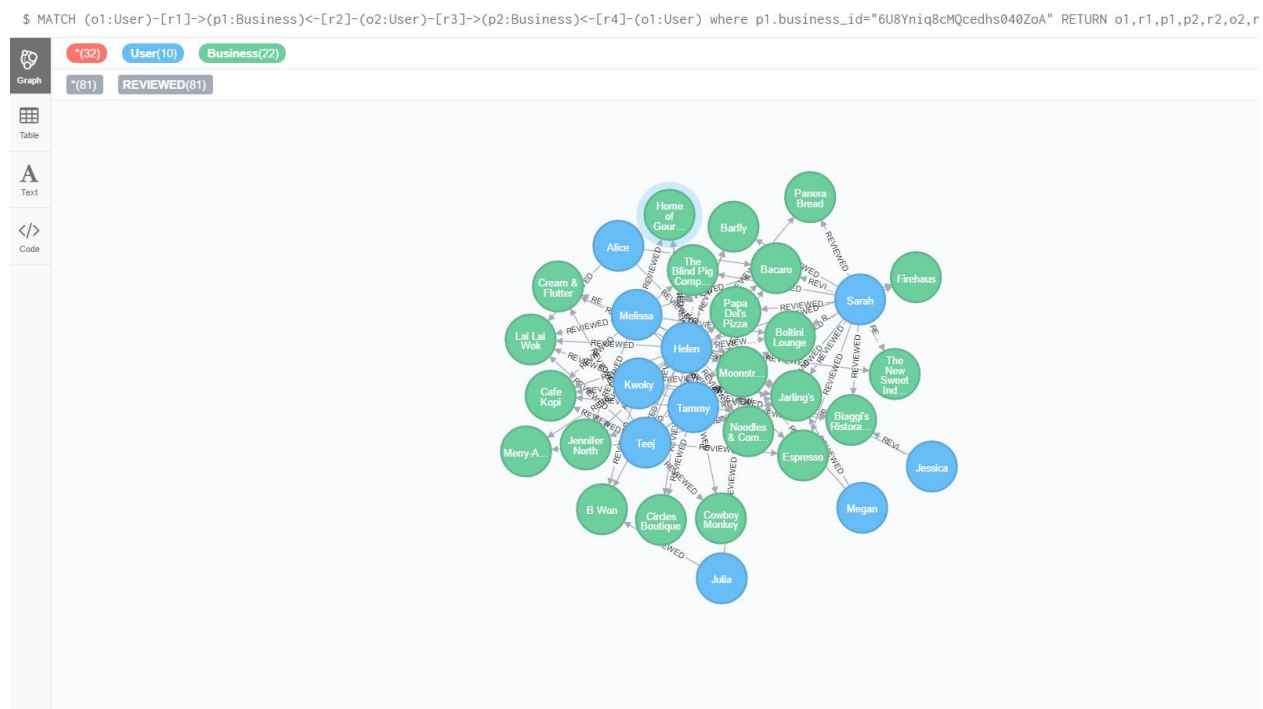


First Query – Reviewers sometimes review as a mob (reviewing the same set of businesses over and over).

```

MATCH (o1:User)-[r1]->(p1:Business)<-[r2]-(o2:User)-[r3]->(p2:Business)<-[r4]-(o1:User)
where p1.business_id="6U8Yniq8cMQcedhs040ZoA"
RETURN o1,r1,p1,p2,r2,o2,r3,r4
LIMIT 50

```



Explanation: For this query, we created a relationship between the business and the user, termed as Review. We are using that relationship here to find the optimised solution. In this, we are representing all the group of users, who reviewed similar businesses, since our User1-Business1-User2-Business2-User1, type connection will give only those pair of Businesses which have at least 2 users in common. We are using a relationship between the nodes User and Business multiple times since we require group of users who have reviewed the same group of Businesses.

Reviewing as a mob behavior is noticed through our query for some businesses but we can only be certain about it once we analyze the complete yelp dataset.

Second Query – Does geographical proximity leads to uniformity in business ratings or divergence?

For Postal Code: 61820

Match (b:Business)

where b.postal_code = "61820"

return stDev(b.stars)/avg(b.stars) as Coefficient_of_Variation



Explanation: In this query, we are calculating the coefficient of Variation(CV) also known as relative standard deviation (RSD). The coefficient of Variation is a matrix which is defined as a measure of relative variability. It is the ratio of the standard deviation to the mean (average). *stDev* and *avg* are mathematical

functions in Neo4j which are used to calculate standard deviation and average of ratings (stars given to each business), respectively. Here we are calculating these only for the postal code 61820. This query gives us the CV. If CV is less than 1, like in our case, it means that the ratings are not much varied in that geographical proximity. Our value of ~0.27 indicates that this region has uniformity in business ratings.

For Postal Code: 61821

Match (b:Business)

where b.postal_code = "61821"

return stDev(b.stars)/avg(b.stars) as Coefficient_of_Variation

\$ Match (b:Business) where b.postal_code = "61821" return stDev(b.stars)/avg(b.stars) as Coefficient_of_Variation	
 Table	Coefficient_of_Variation
 Text	0.297875288191233
 Code	
Started streaming 1 records after 3 ms and completed after 3 ms.	

Explanation: In this query, we are calculating the coefficient of Variation(CV) also known as relative standard deviation (RSD). The coefficient of Variation is a matrix which is defined as a measure of relative variability. It is the ratio of the standard deviation to the mean (average). *stDev* and *avg* are mathematical functions in Neo4j which are used to calculate standard deviation and average of ratings (stars given to each business), respectively. Here we are calculating these only for the postal code 61821. This query gives us the CV. If CV is less than 1, like in our case, it means that the ratings are not much varied in that geographical proximity. Our value of ~0.30 indicates that this region has uniformity in business ratings.

For Postal Code: 61822

Match (b:Business)

where b.postal_code = "61822"

return stDev(b.stars)/avg(b.stars) as Coefficient_of_Variation

\$ Match (b:Business) where b.postal_code = "61822" return stDev(b.stars)/avg(b.stars) as Coefficient_of_Variation	
 Table	Coefficient_of_Variation
 Text	0.29506423272694077
 Code	
Started streaming 1 records after 3 ms and completed after 3 ms.	

Explanation: In this query, we are calculating the coefficient of Variation(CV) also known as relative standard deviation (RSD). The coefficient of Variation is a matrix which is defined as a measure of relative variability. It is the ratio of the standard deviation to the mean (average). *stDev* and *avg* are mathematical functions in Neo4j which are used to calculate standard deviation and average of ratings (stars given to each business), respectively. Here we are calculating these only for the postal code 61822. This query gives us the CV. If CV is less than 1, like in our case, it means that the ratings are not much varied in that geographical proximity. Our value of ~0.29 indicates that this region has uniformity in business ratings.

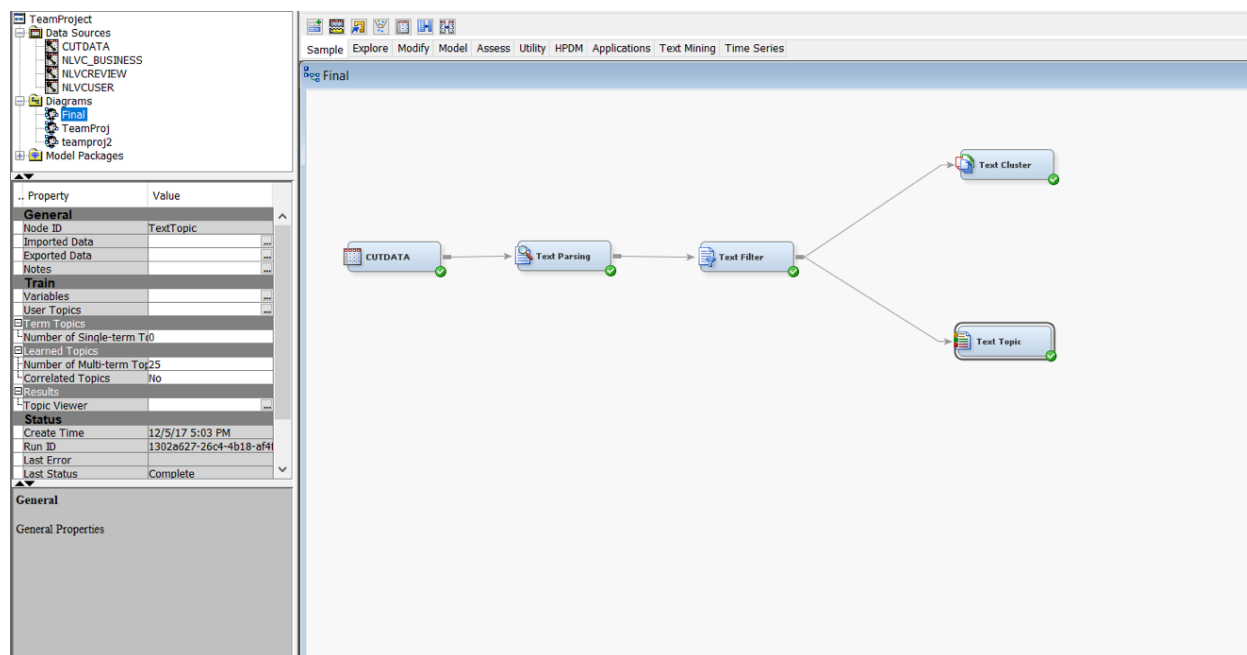
Identifying Fake Reviews with SAS (Statistical Analysis System)

SAS is a software suite that can mine, alter, manage and retrieve data from a variety of sources and perform statistical analysis on it

We are considering a text mining approach to derive a text-based definition for the analysis of fake reviews on Yelp.

Purpose:

Due to the constraint of memory, we are only including relevant data for our processing which includes fields such as Business ID, date, Review_Id, text, user_Id from NVLCReview Dataset.



Text Parsing:

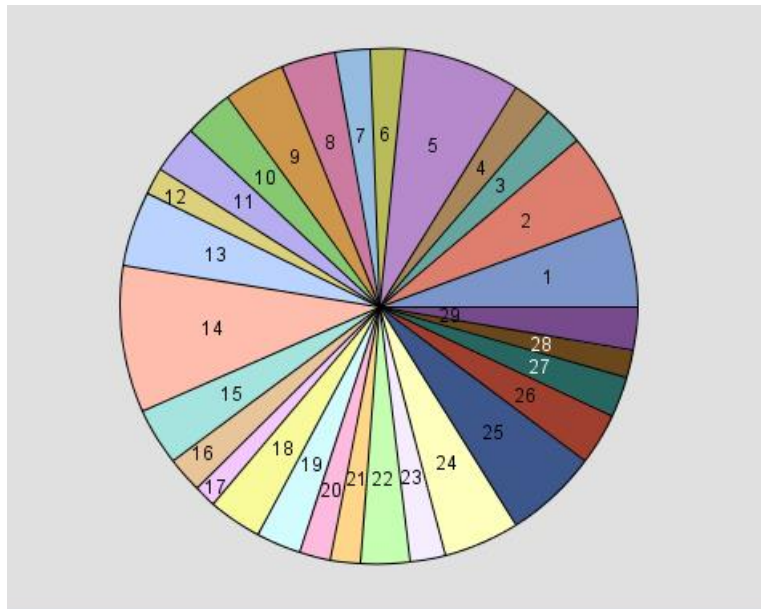
In text parsing, we parsed through the whole dataset and found out the words that were irrelevant for the analysis of reviews. So, in order to process our data, text parsing pallet helped us reduce our dataset by removing those type of words.

Text Filter:

In text filtering, we filtered out all parts of speech except adjectives and defined synonyms for words that have the similar meaning.

Text Cluster:

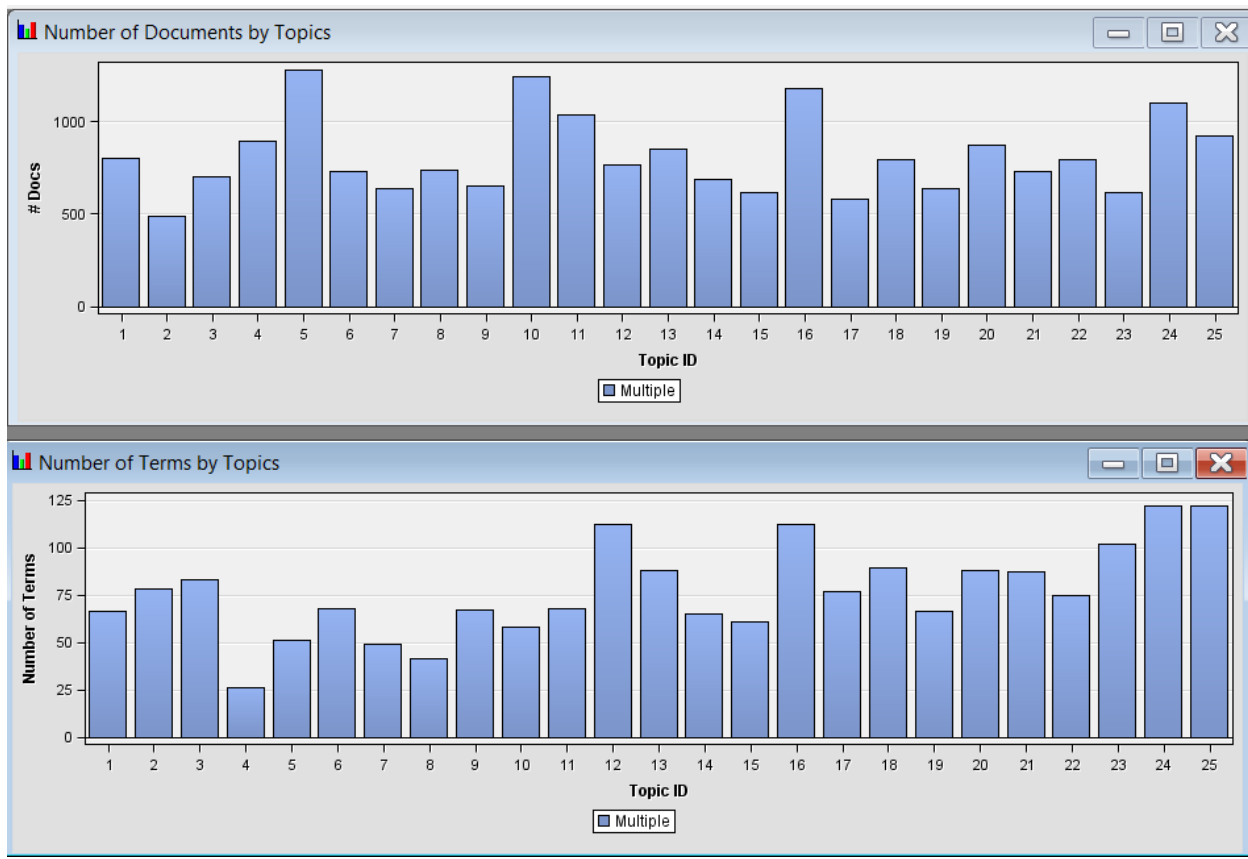
For the whole process, we ran it multiple times till the time all the word with similar meanings formed different sets of cluster



Cluster ID	Descriptive Terms	Frequency	Percentage	Coordinate 1	Coordinate 2	Coordinate 3	Coordinate 4	Coordinate 5	Coordinate 6	Coordinate 7	Coordinate 8	Coordinate 9	Coordinate 10	Coordinate 11
1	pretty decent +sweet 'pretty good' free grilled blue +bland overall +little different chinese art +hard attentive	551	6%	0.295114	0.014647	0.006178	-0.03441	-0.02468	-0.01277	0.067155	-0.02573	0.120955	0.06382	-0.04031
2	delicious red grilled +thin +crispy homemade +light perfect flavorful +soft unique +warm peruvian italian generous	559	6%	0.305971	0.097783	0.018201	-0.12044	-0.02028	-0.08595	-0.03733	-0.00276	-0.00475	-0.00734	-0.04567
3	big +big fan' +big deal' art different home +ferris wheel' audio lucky +small piece' +early 'customer service' +cheap ov	244	2%	0.299903	-0.03009	0.002004	-0.01174	0.003469	-0.02231	0.183616	0.017817	0.04213	0.077426	0.110318
4	open +open bar' open bar option' minute night +ferris wheel' +high roller' half +high afraid able +great view' observation	246	2%	0.279212	-0.45862	-0.03403	0.094655	-0.15783	0.038562	-0.15854	-0.23538	0.233804	-0.27044	-0.08288
5	cheap expensive authentic +cool frozen +hard beautiful +good peruvian +tiny worth super overall italian	741	7%	0.233163	0.010071	-3.45E-5	0.013154	0.002464	-0.02907	0.047965	-0.03349	-0.01913	0.00283	-0.05259
6	blast horrible rude terrible +far +sad different incredible +good thing' +long +long wait' +bad service' polish accommodati	222	2%	0.301138	-0.04302	0.003726	-0.03846	0.167519	0.064428	0.047669	-0.06122	-0.00466	0.100573	-0.03165
7	spicy 'spicy tuna' +spicy tuna roll' dynamite +yummy +dynamite roll' flavorful +crispy +fast +green +sweet fried authen	211	2%	0.307223	0.12575	0.023528	-0.16079	-0.05864	-0.01171	-0.16225	0.095737	0.121157	0.0575	0.035489
8	different next 'a lot of' back several fantastic 'great job' lucky fun +hair cut' +well haircut' 'great place' affordable top glad	328	3%	0.27522	-0.03918	0.004509	-0.04865	0.082511	-0.00744	0.05825	-0.04449	-0.0065	0.050693	-0.02487
9	first +first time' second +first visit' +sally next +busy helpful family peruvian back +bad customer service' +small piece'	386	4%	0.338599	0.011748	0.003477	0.015159	0.503193	0.161353	-0.21638	0.078282	0.003235	-0.1309	0.073705
10	long +large +long line' +large group' +long wait' +young +fast difficult worth +great view' +loud beautiful +hungry obser	322	3%	0.299004	-0.05716	-0.00478	0.028637	0.028482	-0.01601	0.102811	-0.03367	0.042086	-0.00839	0.018231
11	high +high roller' +high end' +ferris wheel' beautiful afraid entire observation fun night +tall minute +great view' half +s	288	3%	0.297279	-0.38585	-0.02594	0.05997	-0.09981	0.072067	0.056946	0.433485	-0.18248	0.064487	-0.0728
12	favorite +favorite place' +favorite restaurant' +favorite spot' +favorite sushi place' +favorite pizza' +favorite thing' fabulous	186	2%	0.24585	0.022351	0.014173	-0.12565	0.031687	-0.09625	-0.04054	-0.00271	-0.07937	-0.02961	-0.06742
13	white frozen +white castle' +fast east west' +east coast' +grocery store' west coast' +cheese slider' +fast food' +soggy	470	5%	0.317525	0.194814	-0.04463	0.46672	-0.03186	-0.05523	-0.05701	0.012871	-0.08179	-0.03609	0.009932
14	able fun +full +busy entire free terrible night +slow rude front +great view' +tall observation 'customer service'	924	9%	0.24586	-0.13564	-0.00629	-0.00322	0.044605	0.033147	0.060787	-0.0456	-0.01696	0.055897	-0.03172
15	fried +fasty 'fried chicken' +polish boy' 'fried rice' polish +hot sauce' +crispy +sweet chinese +hot +juicy peruvian gen	350	4%	0.276682	0.134874	-0.0562	-0.11962	-0.12956	0.115944	-0.12035	0.002467	0.053655	0.072928	-0.04763
16	little pricey +cute difficult +good +salty afraid +tiny expensive +friendly 'great place' +cool +well sushi' art +dirty	212	2%	0.353104	-0.00466	0.002978	-0.0093	-0.03368	-0.04744	0.114901	-0.00406	0.168772	0.095082	0.229767
17	chicken +chicken ring' +chicken slider' +chicken breast' +chicken sandwich' +juicy white fish frozen +cheese slider' +fa	158	2%	0.280475	0.17281	-0.00317	0.140556	-0.12045	0.02964	-0.06426	-0.02414	0.039657	0.062509	-0.04276
18	happy +happy hour' half blue regular professional +cute 'great job' minute +high end' night +great experience' +great p	332	3%	0.289719	-0.20204	-0.0062	-0.04768	-0.00777	-0.24363	-0.23375	-0.20217	-0.28153	0.276406	0.12682
19	small +small place' +small portion' audio +small pizza' +small piece' +friendly super +hard +well sushi' +clean +dyna	260	3%	0.339235	0.067909	0.011808	-0.06768	-0.01849	-0.17169	0.217523	0.019861	0.12615	-0.11575	0.178204
20	bad +bad experience' +bad customer service' +bad pizza' +bad service' +bad review' +bad food' +bad meal' horrible +	192	2%	0.240487	-0.00461	-0.00236	0.05789	0.126124	0.170015	0.147729	-0.03638	0.055544	0.209158	-0.31645
21	huge 'huge fan' dynamite +huge portion' +dynamite roll' local +small 'great place' +favorite restaurant' +small portion' incr	205	2%	0.307517	0.018057	0.009502	-0.06794	0.003599	-0.03181	0.093496	-0.01401	-0.00545	-0.05377	0.033806
22	hot +hot dog' +hot sauce' +hot towel' +cold homemade +great service' reasonable italian +crispy outstanding flavorful	314	3%	0.306289	0.11576	0.023797	-0.16826	-0.15525	0.304424	0.111313	-0.14878	0.31577	-0.21997	0.144893
23	old +hot towel' +young +dirty last free +bad customer service' +sad home +loud +hair cut' front +soggy 'great atmosp	208	2%	0.294975	-0.00364	0.00524	-0.03167	0.064003	0.074046	0.148001	-0.09459	-0.07345	0.076242	-0.0639
24	fresh japanese 'fresh mint' +well sushi' delicious authentic +green flavorful fish +light +friendly accommodating generou	474	5%	0.318112	0.121914	0.024846	-0.16826	-0.03592	-0.20704	-0.0238	0.091021	-0.07239	-0.2086	-0.20233
25	good fantastic +clean +great service' 'great job' knowledgeable +well haircut' +hair cut' beautiful +great experience' gre	602	6%	0.376187	-0.05447	0.012141	-0.14374	0.078947	-0.10591	0.057692	-0.12949	-0.13238	0.013852	-0.12543
26	few professional attentive comfortable +friendly bartender' +early free +young +easy overall +happy awful knowledgeable	320	3%	0.349759	-0.06337	0.003782	-0.04095	0.110869	-0.03675	0.144815	-0.08236	-0.01358	0.090456	-0.07186
27	food +fast +food truck' 'food quality' +fast food' +fast service' +bad food' frozen 'great place' pleasant local +hungry chin	261	3%	0.270941	0.063024	0.005116	0.037493	-0.02738	0.007903	0.084688	-0.05894	-0.01998	0.042562	-0.10067
28		156	2%	0.001152	0.001684	-0.032	-0.00328	0.002335	9.273E-7	0.001639	0.000034	9.501E-5	-1.27E-5	9.575E-6
29	friendly helpful +friendly staff' +friendly service' +good +friendly bartender' knowledgeable +clean attentive professional '	278	3%	0.271219	0.013461	0.01332	-0.1342	0.108942	-0.14521	0.071479	-0.19381	-0.11427	-0.01407	-0.2587

Text Topic:

Text topic helps us explore the document collection by automatically associating terms and documents according to both discovered and user-defined topics.



After we analyzed with text cluster and text topic, we decided to use number of terms and number of documents by topic to determine with our fake review.

We compared the number of terms with number of documents by topics and derived if the number of terms is greater than the number of documents then these kinds of topics are fake reviews.

Web Page Design for Yelp

Location Web Page



For the location page, the user will enter a location on the home page of Yelp and will be redirected to the page above. The metrics on display as part of this page are: -

1. Most Popular Restaurants in the neighbourhood:

As explained in the queries executed in MongoDB, when the user selects a location, that location's coordinates will get passed on to the geospatial query which will find the most popular restaurants within the 5 mile radius of the coordinates provided. The popularity is based on the number of reviews a popular business received sorted in descending order of date.

2. Most Popular Restaurants for delivery in the neighbourhood:

As explained in the queries executed in MongoDB, when the user selects a location, that location's coordinates will get passed on to the geospatial query which will find the most popular restaurants within the 5 mile radius of the coordinates provided. The popularity is based on the number of reviews a popular business received sorted in descending order of date. We will also filter the results on the basis of the business being a restaurant and delivery being provided by that restaurant.

3. Restaurants currently open in the neighbourhood:

As explained in the queries executed in MongoDB, when the user selects a location, that location's coordinates will get passed on to the geospatial query which will find the restaurants within the 5 mile radius of the coordinates provided along with checking whether they are open or not.

Restaurant Web Page



For the restaurant page, the user will select a restaurant and will be redirected to the page above. The metrics on display as part of this page are: -

1. Most Helpful Reviews:

As explained in the text based and non-text based approach to find the helpfulness of a review using MongoDB, we will find the most helpful reviews for a restaurant based on our selected criteria. We also plan to incorporate review comments as discussed in the MongoDB Psuedo Schema which will add extra value towards helpfulness of a review.

2. Attributes of a Restaurant:

In our design above, we have displayed the various attributes that correspond to a restaurant on the page to add more value to a user's decision to visit that restaurant.

Reviews Web Page



For the reviews page, the user will select a restaurant's reviews and will be redirected to the page above. The metrics on display as part of this page are: -

1. Most Recent Reviews:

This is a straight forward query where the most recent reviews will be returned for the user to help him make a calculated decision based on recent experiences of the users.

2. Reviewers with highest average rating:

As explained in the query execution with MongoDB, we were able to retrieve the average ratings of each user based on the reviews posted by them on the complete Yelp setup. These users are sorted in descending order and are displayed on this page for better understanding of the feedback on that business.

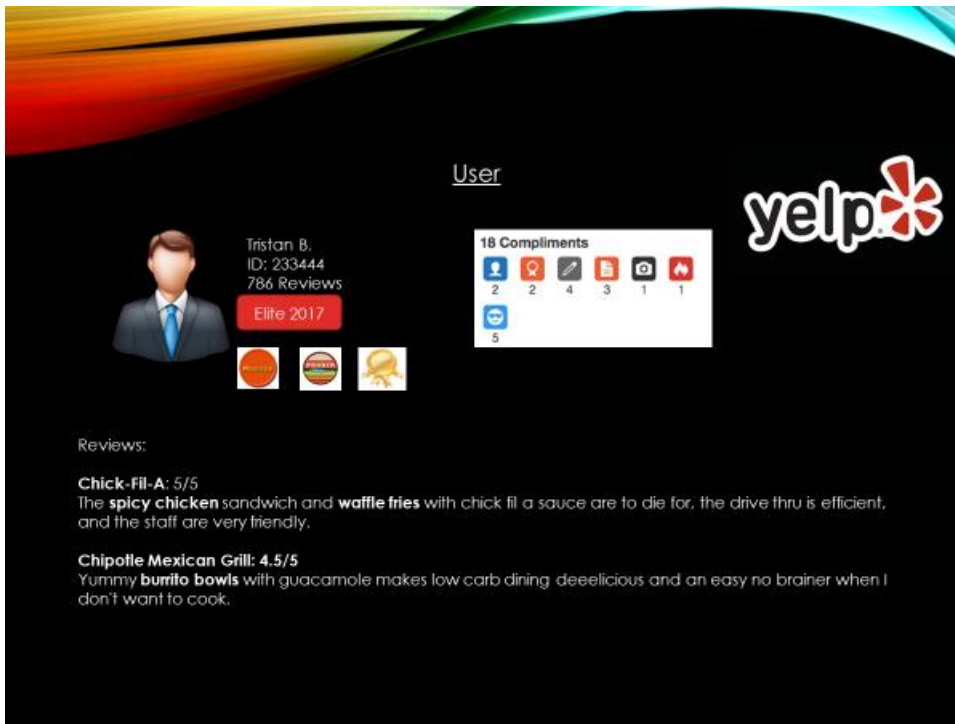
3. Most Prolific Reviewers:

This is a direct query which we executed to find out the users that have posted the maximum number of reviews on Yelp.

4. Oldest Reviewer:

Based on the field `yelping_since`, we could easily figure out the oldest reviewer for a business which add value towards judging the feedback of a restaurant.

User Web Page



For the user page, the user will select another user's profile and will be redirected to the page above. The metrics on display as part of this page are: -

1. Most Helpful Reviews:

As explained in the text based and non-text based approach to find the helpfulness of a review using MongoDB, we will find the most helpful reviews for a user based on our selected criteria. We also plan to incorporate review comments as discussed in the MongoDB Psuedo Schema which will add extra value towards helpfulness of a review.

2. Badges awarded to the User:

Based on our extra addition of the badges document in our MongoDB pseudo schema, we have displayed them on the user's profile to provide them extra incentives and gain better visibility on the Yelp system.