TERM PROJECT

ECE 656 Database Systems

Winter 2018

Group no. 05

Group Members

Samina Islam Eva

UW ID 20741206

Anamika Choudhury

UW ID 20741648

Md Rubayatur Rahim Bhuyian

UW ID 20743420

Department of Electrical & Computer Engineering

Abstract

In this project, an attempt has been made to analyze the Yelp dataset. Yelp dataset is a subset of Yelp's businesses, reviews, and user data. The dataset contains information about businesses across 11 metropolitan areas in four countries. Different analysis on the data has been done and presented in the report. Moreover, several operations were done to understand and prepare the report for analysis such as constructing an E-R diagram, Data cleaning, Data indexing.

Methodology

The project was divided into following steps:

Data Visualization

In this section we tried to understand the data set by plotting several plots.

Data Preparation:

As part of data preparation we did the following checking:

- Consistency checking
- Sanity Checking

These were done to make sure that there are no internal conflicts of the data in the database. Consistency checking were done by several queries and implementing constraints.

Data Indexing

To increase the efficiency of our queries several indexing were done.

Data Analysis

Several analysis on the data has been done to find out interesting findings.

Data Visualizations

Data visualizations were done using python.

Rating Distributions

The below graph represents the ratings and the corresponding number of businesses

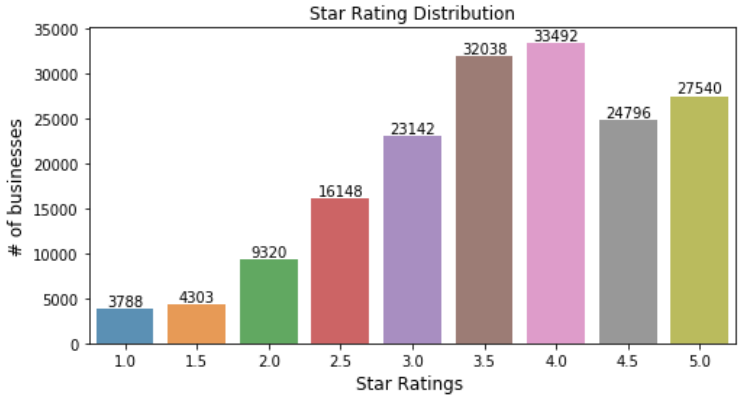


Fig 1: Ratings and the corresponding number of businesses

Popular business categories distribution

The below graph shows the top categories of the business

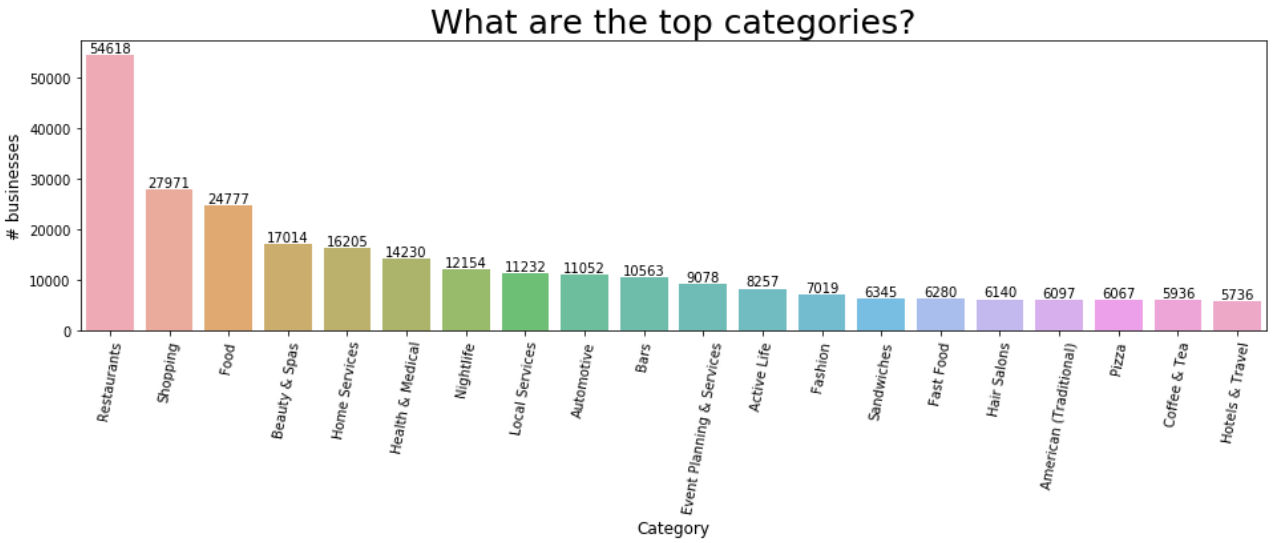


Fig 2: Top categories of the business

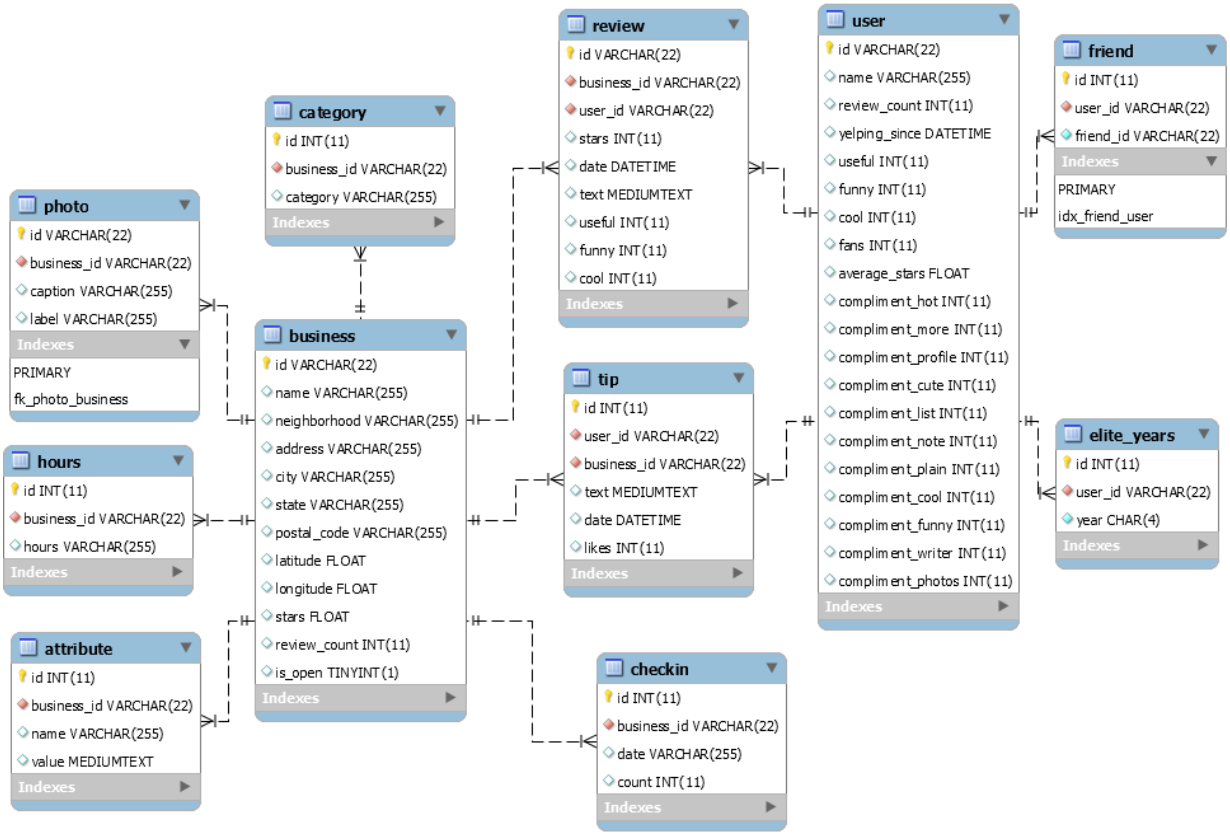
Check in exploration

From below table we can see how many people check in at which time of a day.

Index	hour	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
0	0	178666	163336	224425	225209	171669	156195	166254
11	1	190371	161168	245453	235295	181712	162632	174696
16	2	169989	140252	236195	224716	160906	141732	153923
17	3	127783	106422	189242	184979	117639	103584	111840
18	4	86825	73497	138952	140092	76283	68950	72821
19	5	55672	48253	97062	101133	42130	42977	44109
20	6	36546	31624	68465	72404	27108	26646	27175
21	7	26551	23022	49633	52890	20672	18946	19195
22	8	18004	15327	33591	35079	13830	12452	12484
23	9	12683	11000	21969	21524	10058	8837	8814
1	10	11837	10432	16753	15903	9776	9448	9200
2	11	15617	14580	15833	13724	14791	14397	14568
3	12	26149	23655	21330	16856	24631	24994	24828

4	13	37355	33640	36613	28085	34514	35002	34519
5	14	56684	48243	63294	49652	51012	50415	51139
6	15	79148	66663	103651	86110	70076	67864	69115
7	16	106798	88219	149504	129330	91269	86936	89414
8	17	119295	100422	188682	171205	99254	95791	97497
9	18	156740	131296	225651	205088	131839	126253	128179
10	19	194731	160678	251537	229561	162155	155190	157893
12	20	168226	140018	246969	222193	135046	128641	131959
13	21	154843	127014	229788	197524	124231	117723	120810
14	22	166393	129185	219524	178938	133354	126244	129874
15	23	190530	138087	217466	168674	151170	143508	146355

ER Diagram



Data Preparation

Sanity checking and consistency checking were performed on different tables as mentioned below.

Business Table

1. From this query, we can see that the business table contain 174567 rows means all data are loaded successfully from .sql file.

```
mysql> select count(*) from business;
+----------+
| count(*) |
+----------+
|   174567 |
+----------+
1 row in set (0.05 sec)
```

The id attribute of Business table is valid as there is no empty value or null value so do non-negative. It is assured from the below query.

```
mysql> select count(*) from business where id=''  or id is null or length(id)<1;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.06 sec)
```

We checked all the columns to know which have contained missing or garbage values.

When we checked the “Neighborhood” column we got the below counting:

```
mysql> select count(*) from business where neighborhood is null or neighborhood='';
+----------+
| count(*) |
+----------+
|   106552 |
+----------+
1 row in set (0.06 sec)
```

And for “City”, “Postal Code”, “Address”, ”State”, ”Latitude”, “Longitude” the count for missing values are:

City missing	Postal Code Missing	Address Missing	State, longitude, latitude Missing
+----------+ | count(*) | +----------+ | 1 | +----------+ 1 row in set (0.06 sec)	+----------+ | count(*) | +----------+ | 623 | +----------+ 1 row in set (0.05 sec)	+----------+ | count(*) | +----------+ | 6442 | +----------+ 1 row in set (0.06 sec)	+----------+ | count(*) | +----------+ | 2 | +----------+ 1 row in set (0.06 sec)

User Table

For checking the User Table we at first check whether there is any name is missing or not and found that 495 name is missing so we can say that there are some user who are anonyms.

```
mysql> select count(*) from user where name='';
+----------+
| count(*) |
+----------+
|      495 |
+----------+
1 row in set (0.75 sec)
```

As the yelp started at 2004 so no user can open their account before that and so do a future date cannot be an input for the "yelping_since" attribute. The below query result ensure us that the data was no junk values.

```
mysql> select count(*) from user  where yelping_since<'2004-01-01 00:00:00' or yelping_since>current_date();
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.58 sec)
```

Friend Table

"Friend" table has attributes such as user_id, friend_id. We matched those attributes with "User" table's id to ensure that there was no excess user_id in the friend's table. But for friend_id column contains some counts which indicates that those id are not belong to User table and the reason may be that the user deactivate their account or removed their account. So we can remove those lines for cleaning purpose.

Tip Table

At first checked the user_id of Tip table and there is no junk or empty values. We also checked the unique user_id of Tip with user table and there is no mismatch.

```
mysql> select count(user_id)  from tip left join user on tip.user_id=user.id where user.id is null;
+----------------+
| count(user_id) |
+----------------+
|              0 |
+----------------+
1 row in set (2.08 sec)
```

Similarly, with User table we checked the date column to ensure that no date is below 2004 or above current date.

```
mysql> select count(*) from tip where date<'2004-01-01 00:00:00' or date>current_date();
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.41 sec)
```

Business_id does not contain any empty values and the number of unique business_id is not more than the number of distinct id of Business table. That's why the below query results zero.

```
mysql> select count(*) from tip left join business on tip.business_id=business.id where business.id is null;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (2.76 sec)
```

Review Table

User_id does not contain any empty values and the number of unique user_id is not more than the number of distinct id of User table. That's why the below query results zero.

```
mysql> select count(distinct user_id) from review left join user on review.user_id=user.id where user.id is null;
+--------------------------+
| count(distinct user_id) |
+--------------------------+
|                        0 |
+--------------------------+
1 row in set (21.12 sec)
```

Similarly, for Business_id we got the below result and from the result it ensures us that there is no mismatch.

```
mysql> select count(distinct business_id) from review left join business on review.business_id=business.id where business.id is null;
+----------------------------+
| count(distinct business_id) |
+----------------------------+
|                          0 |
+----------------------------+
1 row in set (28.16 sec)
```

Attributes Table

We checked the attributes name as well as values to confirm whether there is any null value or not.

```
mysql> select count(*) from attribute where name is null or length(name)<1;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.43 sec)

mysql> select count(*) from attribute where value is null or length(value)<1;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (1.28 sec)
```

Category Table

We checked the category name to check whether there is any null value or not. And the count shows zero.

```
mysql> select count(*) from category where category is null or length(category)<1;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.22 sec)
```

Similarly, checked the business_id.

```
mysql> select count(distinct business_id) from category left join business on category.business_id=business.id where business.id is null;
+----------------------------+
| count(distinct business_id) |
+----------------------------+
|                          0 |
+----------------------------+
1 row in set (1.74 sec)
```

Checkin Table

The query shows that all the date attributes of checkin table have valid values.

```
mysql> select count(*) from checkin where date is null or date='';
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (1.19 sec)
```

Hours Table

As all the business_id of Hours table must contains those which have in Business Table. And the query shows that no excess entry is in Hours table for business_id.

```
mysql> select count(distinct business_id) from hours left join business on hours.business_id=business.id where business.id is null;
+----------------------------+
| count(distinct business_id) |
+----------------------------+
|                          0 |
+----------------------------+
1 row in set (1.99 sec)
```

Check constraint add

We have added some constraints so there is no scope to add junk values there.

For Business Table

The system will check before inserting any data into “review_count”,” stars” attributes to ensure that the values must be positive.

And for “is_open” the value must be boolean, so it will insert those values which is 0 or 1.

```
mysql> alter table business add constraint b_reviewCount check (review_count>=0);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table business add constraint b_stars check (stars>=0);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table business add constraint b_isopen check (is_open in(0,1));
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

For User Table

We check the review_count just like Business table. Besides this, for yelping_since we check that the time must be equal greater than 2004. So, we add a constraint, so it can check the value before insertion.

```
mysql> alter table user add constraint u_reviewCount check (review_count>=0);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table user add constraint u_yelpingSince check (yelping_since>='2004-01-01 00:00:00');
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

In User table there are lots of attributes which are numeric so for each of them we add constraints so that it doesn't insert any non-negative values.

Similarly, for Tip table the date has to be checked so that it cannot insert any time before 2004, Review table check all the numeric columns so every inserted values must be positive.

Additional checking (described in project)

Elite_years consistency check

One user cannot be elite before they started reviewing. The count value shows that it meets the requirement

```
mysql> select count(*) from elite_years join user on user.id=elite_years.user_id  where
year<SUBSTRING(yelping_since, 1, instr(yelping_since,'-')-1);
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.35 sec)
```

Yelp started on 2004 so a user cannot be elite before 2004 and so do the years cannot include any year above current time. From the below query we can say that the data is consistent.

```
mysql> select count(*) from elite_years where year<2004 or year>2017;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.07 sec)
```

The number of reviews written by a user as recorded in the User table cannot be smaller than the number of reviews in the sample set.

From our findings it is not consistent. There are 1323 entries for which the reviews of sample set for a user is larger than the number of reviews given by that user.

```
mysql> select count(*) from (
    -> select count(review.id) sample_review,user.review_count userReview from review right join user
    -> on user.id=review.user_id group by user.id)a
    -> where a.userReview<a.sample_review;
+----------+
| count(*) |
+----------+
|     1323 |
+----------+
1 row in set (22.02 sec)
```

For Example:

For a user (among 1323) we get the following result:

User Table:

```
mysql> select id as user_id,review_count from user where id='-61V4ZkRsKUChYFZtdZDvQ';
+------------------------+--------------+
| user_id                | review_count |
+------------------------+--------------+
| -61V4ZkRsKUChYFZtdZDvQ |            0 |
+------------------------+--------------+
1 row in set (0.00 sec)
```

Review Table (sample set):

```
mysql> select user_id,count(id)review_count from review where user_id='-61V4ZkRsKUChYFZtdZDvQ';
+------------------------+--------------+
| user_id                | review_count |
+------------------------+--------------+
| -61V4ZkRsKUChYFZtdZDvQ |            4 |
+------------------------+--------------+
```

Consistency check for the ratings from the Review table vs the ratings from the Business Table

The data is consistent. Let's say, 0.6 is the threshold level. So, there are 63 entries present which has a difference more than 0.6

```
mysql> select count(*) from (
    -> select b.id,b.stars as businessOverallStars, avg(r.stars) as userGivenStars from business b  left join review r
    -> on b.id=r.business_id  where business_id is not null group by business_id)a where ABS(a.businessOverallStars-a.userGivenStars)>0.6;
+----------+
| count(*) |
+----------+
|       63 |
+----------+
```

Consistency check for that no business is open more than 7days

From our findings the data is consistent.

```
mysql> select count(distinct category) from(
    -> select category, count( distinct SUBSTRING(hours, 1, instr(hours,'|')-1) )AS days from business b join  hours h
    -> on h.business_id=b.id join category c on  c.business_id=b.id group by category )a where a.days >7;
+--------------------------+
| count(distinct category) |
+--------------------------+
|                        0 |
+--------------------------+
1 row in set (2 min 51.71 sec)
```

Indexing

All business_id and user_id of every table has been added as foreign key references business and user table respectively. For example:

```
mysql> ALTER TABLE tip ADD constraint fk_tip_business FOREIGN KEY (business_id) REFERENCES business(id);
Query OK, 1098325 rows affected (22.66 sec)
Records: 1098325  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE tip ADD constraint fk_tip_user FOREIGN KEY (user_id) REFERENCES user(id);
Query OK, 1098325 rows affected (33.28 sec)
Records: 1098325  Duplicates: 0  Warnings: 0
```

In Mysql all the primary key and foreign key are auto index. So, we don't need to declare any kind of indexing for primary and foreign keys. Besides this, we can add some indexing for the most frequent searching attributes. For example, yelping_since as we can search a user based on date.

Indexing mainly depends on which attributes we use mostly for any kind of searching so there are no fixed rules for indexing.

Before index set up:

```
mysql> select count(distinct user_id) from friend;
+--------------------------+
| count(distinct user_id) |
+--------------------------+
|                   760008 |
+--------------------------+
1 row in set (1 min 5.78 sec)
```

After index set up:

```
mysql> select count(distinct user_id) from friend;
+--------------------------+
| count(distinct user_id) |
+--------------------------+
|                   760008 |
+--------------------------+
1 row in set (3.87 sec)
```

Analysis of the dataset

Based on the data, determine if a business is declining or improving in its ratings:

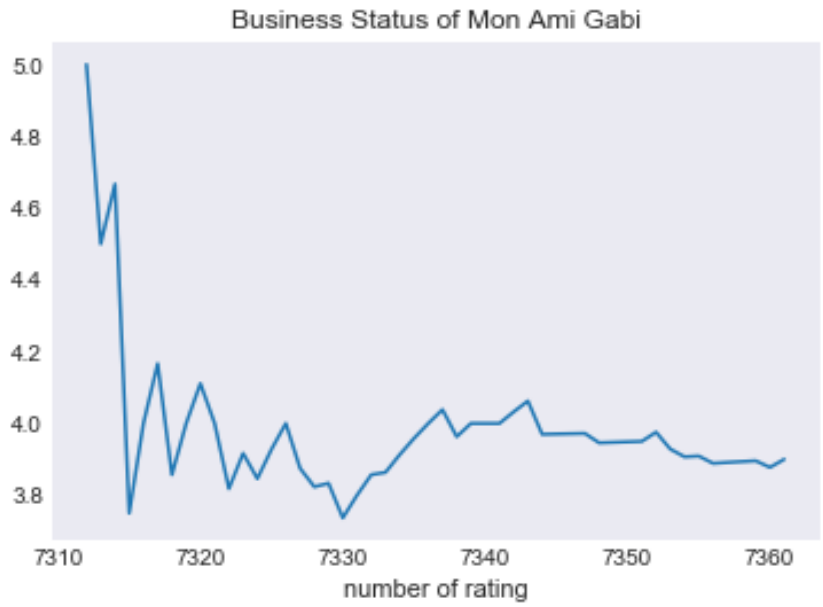
To know whether the business is improving or not we count all the stars of a particular business of all year and then we calculate the latest one-year stars of that business and compare it. If the latest years stars are high than the overall years rating, then it means the business is improving otherwise not. The below query shows that results.

```
mysql> select a.business_id,totalStars,latestStars , case
    -> when latestStars>totalStars then "Improving Business"
    -> when latestStars<totalStars then "Declining Business"
    -> else "Business Stable"
    -> end 'BusinessStatus' from
    -> (select r.business_id ,count(r.id) totalCount,avg(r.stars) totalStars
    -> from review r group by business_id limit 200) a
    -> join
    -> (select r1.business_id ,count(r1.id) latestCount,avg(r1.stars) latestStars
    -> from review r1  where r1.date between date_sub(current_date(),interval 1 YEAR) and current_date()
    -> group by business_id limit 200)b  on a.business_id=b.business_id where abs(latestStars-totalStars)>0 and a.totalCount>50;
+------------------------+------------+-------------+--------------------+
| business_id            | totalStars | latestStars | BusinessStatus     |
+------------------------+------------+-------------+--------------------+
| --9e1ONYQuAa-CB_Rrw7Tw |     4.0889 |      4.2157 | Improving Business |
| --cjBEbXMI2obtaRHNSFrA |     3.1346 |      3.5000 | Improving Business |
| --cZ6Hhc9F7VkKXxHMVZSQ |     3.9528 |      3.9375 | Declining Business |
| --FBCX-N37CMYDfs790Bnw |     3.7222 |      2.7692 | Declining Business |
| --I7YYLada0tSLkORTHb5Q |     3.4861 |      3.6471 | Improving Business |
| --ujyvoQlwVoBgMYtADiLA |     3.6912 |      4.1429 | Improving Business |
| -01XupAWZEXbdNbxNg5mEg |     2.8571 |      2.7500 | Declining Business |
| -050d_XIor1NpCuWkbIVaQ |     3.9871 |      1.0000 | Declining Business |
| -092wE7j5HZOogMLAh40zA |     3.3494 |      5.0000 | Improving Business |
| -0NrB58jqKqJfuUCDupcsw |     3.3412 |      2.7895 | Declining Business |
| -0qht1roIqleKiQkBLDkbw |     2.9938 |      2.9697 | Declining Business |
| -0t6o9LXMVQ3aV8CHhRTnA |     3.3810 |      2.9375 | Declining Business |
| -0tgMGl7D9B10YjSN2ujLA |     3.3167 |      3.1667 | Declining Business |
| -0WegMt6Cy966qlDKhu6jA |     2.1429 |      1.8462 | Declining Business |
| -11PbySWhJQtK6USx4IP2A |     4.1031 |      4.0513 | Declining Business |
| -1UMR00eXtwaeh59pEiDjA |     3.4400 |      2.9014 | Declining Business |
| -1VaIJza42Hjev6ukacCNg |     3.8466 |      3.7500 | Declining Business |
| -1xuC540Nycht_iWFeJ-dw |     4.2425 |      4.1493 | Declining Business |
| -2pQf1ceDZyE2ReCNbj-3A |     3.4630 |      3.4000 | Declining Business |
| -2qfXy3hcWG0GEcE8QUKQw |     2.5469 |      1.9286 | Declining Business |
| -2ToCaDFpTNmmg3QFzxcWg |     1.5011 |      1.3571 | Declining Business |
+------------------------+------------+-------------+--------------------+
```

From the below query, it shows how many companies are declining their business based on ratings.

```
mysql> select count(*)  from
    -> (select r.business_id ,count(r.id) totalCount,avg(r.stars) totalStars
    -> from review r group by business_id ) a
    -> join
    -> (select r1.business_id ,count(r1.id) latestCount,avg(r1.stars) latestStars
    -> from review r1  where r1.date between date_sub(current_date(),interval 1 YEAR) and current_date()
    -> group by business_id )b  on a.business_id=b.business_id where latestStars<totalStars;
+----------+
| count(*) |
+----------+
|    44699 |
+----------+
1 row in set (5 min 48.61 sec)
```

From python we plot graph of a particular business named ‘Mon Ami Gabi’ and from that we can say that the business is declining according to ‘stars rating’.



Does their any match between the ratings given by a user vs the business have its rating? And what is the percentage of the prediction a user given reviews matches with the original business review?

We took a random user and from their given reviews, fetch the respective business_id list.

And then count the distinct id (excluding some of their business id for future evaluation) and it is 107.

```
mysql> select count(business_id)
    -> from review r
    -> join business b
    -> on r.business_id=b.id
    -> where user_id='---1lKK3aKOuomHnwAkAow'
    -> and business_id not in
    -> ('XJGMgs9Kh4kcgf8Oskiewg',
    -> 'cHuA0Yb5oYwx1lrNVABqdQ',
    -> 'bPcqucuuClxYrIM8xWoArg',
    -> 'rq5dgoksPHkJwJNQKlGQ7w',
    -> 'R-McIj4Psxl1VlEacsXeRg',
    -> 'Vg1C_1eqwIwkZLIXGMTW3g',
    -> '5cbsjFtrntUAeUx51FaFTg',
    -> 'kosTPb88O4Q0XGbVbEOGCA',
    -> 'yp2nRId4v-bDtrYl5A3F-g',
    -> 'sZsJooAzpKqOvDysphkqpQ');
+--------------------+
| count(business_id) |
+--------------------+
|                107 |
+--------------------+
```

Then get the count where the difference of stars of a business and the overall reviews of that business is less than 1.5 and it is 77.

```
mysql> select count(business_id)
    -> from review r
    -> join business b
    -> on r.business_id=b.id
    -> where user_id='---1lKK3aKOuomHnwAkAow'
    -> and business_id not in
    -> ('XJGMgs9Kh4kcgf8Oskiewg',
    -> 'cHuA0Yb5oYwx1lrNVABqdQ',
    -> 'bPcqucuuClxYrIM8xWoArg',
    -> 'rq5dgoksPHkJwJNQKlGQ7w',
    -> 'R-McIj4Psxl1VlEacsXeRg',
    -> 'Vg1C_1eqwIwkZLIXGMTW3g',
    -> '5cbsjFtrntUAeUx51FaFTg',
    -> 'kosTPb88O4Q0XGbVbEOGCA',
    -> 'yp2nRId4v-bDtrYl5A3F-g',
    -> 'sZsJooAzpKqOvDysphkqpQ')
    -> and abs(r.stars-b.stars)<1.5;
+--------------------+
| count(business_id) |
+--------------------+
|                 77 |
+--------------------+
1 row in set (0.00 sec)
```

So, we can say that the prediction level is 72%.

The below query shows some result of some business_id of that user with their prediction percentage.

```
mysql> select distinct business_id ,b.stars as businessReview,r.stars as userReview,
    -> case
    -> when r.stars<b.stars then CONCAT(cast((r.stars/b.stars)*100 as decimal(10,2)),'%')
    -> when b.stars<r.stars then CONCAT(cast((b.stars/r.stars)*100 as decimal(10,2)),'%')
    -> else CONCAT((b.stars/r.stars)*100,'%')
    -> end prediction
    -> from review r
    -> join business b
    -> on r.business_id=b.id
    -> where user_id='---1lKK3aKOuomHnwAkAow'
    -> and business_id not in
    -> ('XJGMgs9Kh4kcgf8Oskiewg',
    -> 'cHuA0Yb5oYwx1lrNVABqdQ',
    -> 'bPcqucuuClxYrIM8xWoArg',
    -> 'rq5dgoksPHkJwJNQKlGQ7w',
    -> 'R-McIj4Psxl1VlEacsXeRg',
    -> 'Vg1C_1eqwIwkZLIXGMTW3g',
    -> '5cbsjFtrntUAeUx51FaFTg',
    -> 'kosTPb88O4Q0XGbVbEOGCA',
    -> 'yp2nRId4v-bDtrYl5A3F-g',
    -> 'sZsJooAzpKqOvDysphkqpQ')
    -> and abs(r.stars-b.stars)<1.5;
+------------------------+----------------+------------+------------+
| business_id            | businessReview | userReview | prediction |
+------------------------+----------------+------------+------------+
| hubbaEcYPYEZu5Ziz6i0lw |            3.5 |          4 | 87.50%     |
| YbKjkJCD3lcQcLSMNKglKg |              5 |          5 | 100%       |
| 2Cs9bSN-fMnY3H-0pFP1mg |              4 |          5 | 80.00%     |
| mz9ltimeAIy2c2qf5ctljw |            4.5 |          5 | 90.00%     |
| RRw9I8pHt5PzgYGT2QeODw |              4 |          5 | 80.00%     |
| gHoP4eJimaMltfUlp6EP4Q |              4 |          4 | 100%       |
| qmymSqVwHYRqdwfcBatzpQ |              4 |          5 | 80.00%     |
| XjP43Fso4cL9pFrYl738sg |              4 |          5 | 80.00%     |
| VmssDHtnBjYJQ6cWiwV9kA |            4.5 |          4 | 88.89%     |
| uKOZw2DpJHKuZjozWMCOAw |              4 |          5 | 80.00%     |
| w-As0KSwy8pqMClOea-NLQ |            4.5 |          4 | 88.89%     |
| y8d90Pt16Nip-B5UXWBP-w |              4 |          4 | 100%       |
| iPM85PQMs7QoAxw-Og9ChQ |            3.5 |          3 | 85.71%     |
| ZgPnRzWjQR5NtiauGBww7g |              4 |          5 | 80.00%     |
| Ks0M3J4vZAKsHPuCINz5fQ |            3.5 |          4 | 87.50%     |
| MQiNywdecInMlkW06WYaCg |            3.5 |          4 | 87.50%     |
| hk4NPN6W2xt30Cp-Fiqtuw |            3.5 |          4 | 87.50%     |
| jobP3ywRd3QNZ_GCoPG2DQ |            3.5 |          3 | 85.71%     |
```

Then we do it for those part which we excluded initially and got the prediction. From the below results we can say that our hypothesis is right.

```
mysql> select distinct business_id ,b.stars as businessReview,r.stars as userReview,
    -> case
    -> when r.stars<b.stars then CONCAT(cast((r.stars/b.stars)*100 as decimal(10,2)),'%')
    -> when b.stars<r.stars then CONCAT(cast((b.stars/r.stars)*100 as decimal(10,2)),'%')
    -> else CONCAT((b.stars/r.stars)*100,'%')
    -> end prediction, abs(r.stars-b.stars) as Difference
    -> from review r
    -> join business b
    -> on r.business_id=b.id
    -> where user_id='---1lKK3aKOuomHnwAkAow'
    -> and business_id  in
    -> ('XJGMgs9Kh4kcgf8Oskiewg',
    -> 'cHuA0Yb5oYwx1lrNVABqdQ',
    -> 'bPcqucuuClxYrIM8xWoArg',
    -> 'rq5dgoksPHkJwJNQKlGQ7w',
    -> 'R-McIj4Psxl1VlEacsXeRg',
    -> 'Vg1C_1eqwIwkZLIXGMTW3g',
    -> '5cbsjFtrntUAeUx51FaFTg',
    -> 'kosTPb88O4Q0XGbVbEOGCA',
    -> 'yp2nRId4v-bDtrYl5A3F-g',
    -> 'sZsJooAzpKqOvDysphkqpQ')
    -> and abs(r.stars-b.stars)<1.5;
+------------------------+----------------+------------+------------+------------+
| business_id            | businessReview | userReview | prediction | Difference |
+------------------------+----------------+------------+------------+------------+
| XJGMgs9Kh4kcgf8Oskiewg |            4.5 |          5 | 90.00%     |        0.5 |
| cHuA0Yb5oYwx1lrNVABqdQ |              4 |          5 | 80.00%     |          1 |
| bPcqucuuClxYrIM8xWoArg |              4 |          5 | 80.00%     |          1 |
| rq5dgoksPHkJwJNQKlGQ7w |              4 |          5 | 80.00%     |          1 |
| R-McIj4Psxl1VlEacsXeRg |              4 |          4 | 100%       |          0 |
| kosTPb88O4Q0XGbVbEOGCA |              4 |          4 | 100%       |          0 |
| sZsJooAzpKqOvDysphkqpQ |              4 |          5 | 80.00%     |          1 |
+------------------------+----------------+------------+------------+------------+
7 rows in set (0.01 sec)
```

Do operation hours affect the rating of a business?

We divided the days into two parts. Working days and Weekend. Then we get the average stars for working days and weekends and we can see that the average stars for overall working days are better compare to weekends.

WeekDays

```
mysql> select avg(stars) from(
    -> select b.id,stars from business b
    -> join hours h on h.business_id=b.id
    -> where b.id  not in (select business_id from hours h where hours  like 'satur%' or hours  like 'sun%')
    -> group by b.id)a ;
+-------------------+
| avg(stars)        |
+-------------------+
| 3.9678536475193855 |
+-------------------+
```

WeekEnds

```
mysql> select avg(stars) from(
    -> select b.id,stars from business b
    -> join hours h on h.business_id=b.id
    -> where b.id in (select business_id from hours h where hours  like 'satur%' or hours  like 'sun%')
    -> group by b.id)a ;
+-------------------+
| avg(stars)        |
+-------------------+
| 3.735861207795017 |
+-------------------+
```

Among Saturday and Sunday, Saturday's rating Is better.

```
mysql> select days,avg(stars) 'average stars' from (
    -> select distinct(id) ,SUBSTRING(hours, 1, instr(hours,'|')-1) AS days,stars from
    -> business b join  hours h on h.business_id=b.id
    -> where hours  like 'satur%' or hours  like 'sun%')a group by days;
+----------+--------------------+
| days     | average stars      |
+----------+--------------------+
| Saturday | 3.7338830065941018 |
| Sunday   |  3.656195102475379 |
+----------+--------------------+
2 rows in set (1 min 19.54 sec)
```

But it is not a perfect way to evaluate as it depends on the category of business that which on will work better on weekdays and which will work better on weekends. So most generic way is to evaluate it based on business type. For example, considering the “advertising” category for both weekdays and weekends.

Weekdays

```
mysql> select avg(stars) from(
    -> select b.id,stars from business b
    -> join hours h on h.business_id=b.id
    -> join category on b.id=category.business_id
    -> where category like '%Advertising%' and  b.id  not in (select business_id from hours h where hours  like 'satur%' or hours  like 'sun%')
    -> group by b.id)a ;
+------------------+
| avg(stars)       |
+------------------+
| 4.185840707964601 |
+------------------+
```

Weekends

```
mysql> select avg(stars) from(
    -> select b.id,stars from business b
    -> join hours h on h.business_id=b.id
    -> join category on b.id=category.business_id
    -> where category like '%Advertising%' and  b.id   in (select business_id from hours h where hours  like 'satur%' or hours  like 'sun%')
    -> group by b.id)a ;
+------------------+
| avg(stars)       |
+------------------+
| 4.521739130434782 |
+------------------+
```

From the above result we can say that, “advertising” get more stars on weekends hours compare to weekdays. As most people go through this on holidays.

Similarly, for “restaurants” type the stars depends on hours which include weekdays as mostly people have there after office or in between office.

Weekends

```
mysql> select category,avg(stars) from(
    -> select category,b.id,stars from business b
    -> join hours h on h.business_id=b.id
    -> join category on b.id=category.business_id
    -> where category like 'Restaurants%' and  b.id   in (select business_id from hours h where hours  like 'satur%' or hours  like 'sun%')
    -> group by b.id)a
    -> group by category ;
+-------------+------------------+
| category    | avg(stars)       |
+-------------+------------------+
| Restaurants | 3.562067462263637 |
+-------------+------------------+
```

Weekdays

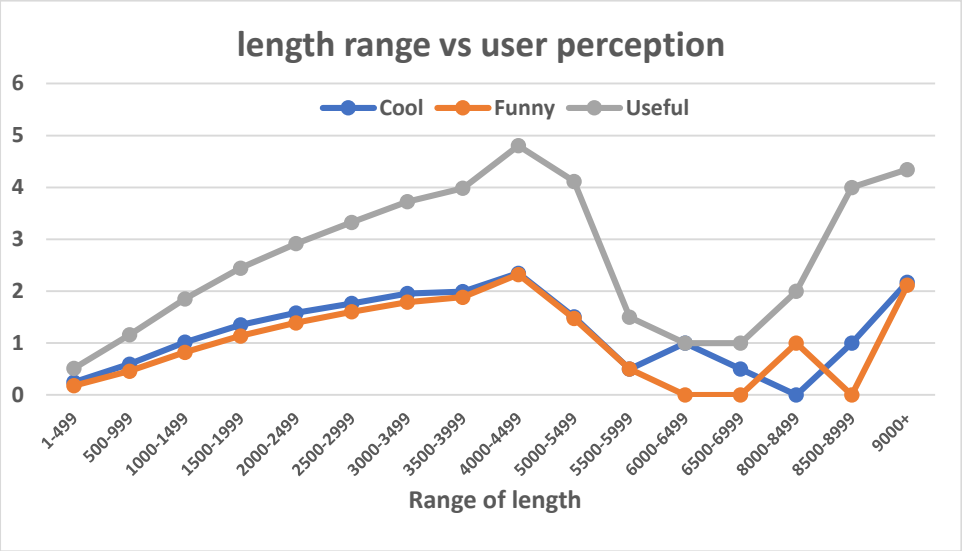
```
mysql> select category,avg(stars) from(
    -> select category,b.id,stars from business b
    -> join hours h on h.business_id=b.id
    -> join category on b.id=category.business_id
    -> where category like 'Restaurants%' and  b.id  not in (select business_id from hours h where hours  like 'satur%' or hours  like 'sun%')
    -> group by b.id)a
    -> group by category ;
+-------------+------------------+
| category    | avg(stars)       |
+-------------+------------------+
| Restaurants | 3.771604938271605 |
+-------------+------------------+
```

Does review length affect how other users perceive a review?

For this we need to get the length of text and the max length is more than 9000. So, for better visualization we divide the length into some ranges. We took first 500 chars into one range than next 500 in another range and so on. And after that we use the average of cool, useful and funny to know is there any relation between the length of a user comment and what other thinks about that.

```
mysql> SELECT CASE WHEN length(text) < 500 THEN '1-499'
    -> WHEN length(text) >= 500 AND length(text) < 1000 THEN '500-999'
    -> WHEN length(text) >= 1000 AND length(text) < 1500 THEN '1000-1499'
    -> WHEN length(text) >= 1500 AND length(text) < 2000 THEN '1500-1999'
    -> WHEN length(text) >= 2000 AND length(text) < 2500 THEN '2000-2499'
    -> WHEN length(text) >= 2500 AND length(text) < 3000 THEN '2500-2999'
    -> WHEN length(text) >= 3000 AND length(text) < 3500 THEN '3000-3499'
    -> WHEN length(text) >= 3500 AND length(text) < 4000 THEN '3500-3999'
    -> WHEN length(text) >= 4000 AND length(text) < 4500 THEN '4000-4499'
    -> WHEN length(text) >= 4500 AND length(text) < 5000 THEN '4500-4999'
    -> WHEN length(text) >= 5000 AND length(text) < 5500 THEN '5000-5499'
    -> WHEN length(text) >= 5500 AND length(text) < 6000 THEN '5500-5999'
    -> WHEN length(text) >= 6000 AND length(text) < 6500 THEN '6000-6499'
    -> WHEN length(text) >= 6500 AND length(text) < 7000 THEN '6500-6999'
    -> WHEN length(text) >= 7000 AND length(text) < 7500 THEN '7000-7499'
    -> WHEN length(text) >= 7500 AND length(text) < 8000 THEN '7500-7999'
    -> WHEN length(text) >= 8000 AND length(text) < 8500 THEN '8000-8499'
    -> WHEN length(text) >= 8500 AND length(text) < 9000 THEN '8500-8999'
    -> WHEN length(text) >= 9000 AND length(text) < 9500 THEN '9000-9499'
    -> ELSE '9500+'
    -> END AS TextLengthRange,avg(cool),avg(funny),avg(useful)
    -> FROM  review group by TextLengthRange;
+-----------------+-----------+------------+-------------+
| TextLengthRange | avg(cool) | avg(funny) | avg(useful) |
+-----------------+-----------+------------+-------------+
| 1-499           |    0.2454 |     0.1797 |      0.5123 |
| 1000-1499       |    1.0189 |     0.8229 |      1.8525 |
| 1500-1999       |    1.3517 |     1.1387 |      2.4452 |
| 2000-2499       |    1.5812 |     1.3898 |      2.9167 |
| 2500-2999       |    1.7651 |     1.6045 |      3.3284 |
| 3000-3499       |    1.9547 |     1.7899 |      3.7242 |
| 3500-3999       |    1.9912 |     1.8834 |      3.9833 |
| 4000-4499       |    2.1731 |     2.1166 |      4.3464 |
| 4500-4999       |    2.3484 |     2.3215 |      4.8063 |
| 500-999         |    0.5959 |     0.4595 |      1.1596 |
| 5000-5499       |    1.5036 |     1.4746 |      4.1162 |
| 5500-5999       |    0.5000 |     0.5000 |      1.5000 |
| 6000-6499       |    1.0000 |     0.0000 |      1.0000 |
| 6500-6999       |    0.5000 |     0.0000 |      1.0000 |
| 8000-8499       |    0.0000 |     1.0000 |      2.0000 |
| 9000-9499       |    1.0000 |     0.0000 |      4.0000 |
+-----------------+-----------+------------+-------------+
```

Following graph is obtained



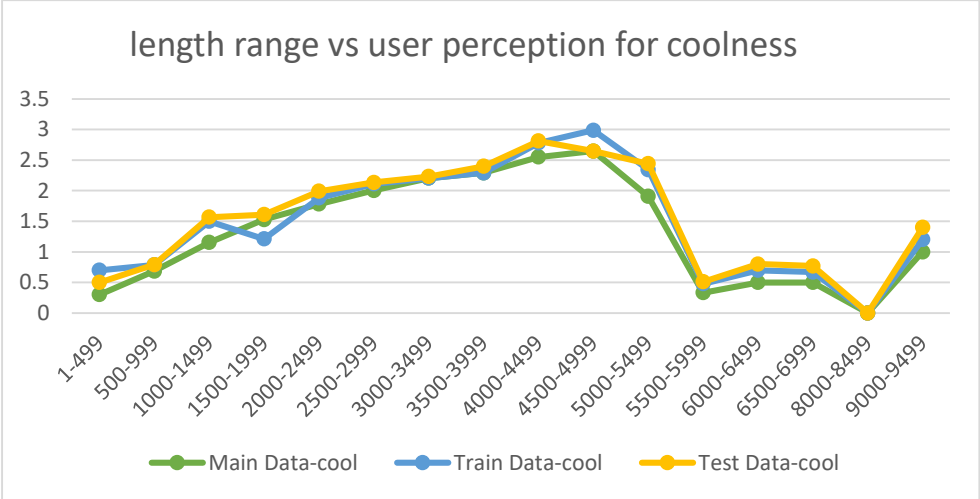
From the above graph we can say that from 1 to 4999 the user perception is increasing compared to length. After that it decrease, so we can say after 5000 length the users don't take seriously the reviews. It may be junk comment or garbage. But after 9000 it attracts the user to read the comment and give the comment a high vote (funny, cool, useful)

For validating our approach, we divided the data into two set one is train and other is test by using the following query:

insert into review_train select * from review order by rand() limit 2630834;

insert into review_test select * from review order by rand() limit 2630834;

Then we re validate user perception on coolness and we find the below graph:



from the graph we can say that for 'COOL' attribute the variation is less among "review_train", "review_test" and "review" (main table without partition)

Part 2

For this we created five users.

1. A casual user is the one who can access it from anywhere and who doesn't want any account and even they cannot write a review they can only uses the application to browse search results.

```
mysql> Create view searchResults as
    -> select b.name,b.address,b.city,b.postal_code,b.stars as business_stars,r.text,r.date ,r.stars as review_stars from business b join
    -> review r on r.business_id=b.id;
Query OK, 0 rows affected (0.02 sec)

mysql> create user ''@'%' ;
Query OK, 0 rows affected (0.05 sec)

mysql> GRANT select  ON yelp_db.search TO  ''@'%';
Query OK, 0 rows affected (0.00 sec)
```

For this we created a view search where we added some specific columns which the casual users can browse.

In the above query we created an user.

create user "@'%' ; here '' means that any user can search it, we don't need to specify it and '%' means that from any host they can browse it.

2. A critiques can search the result and so give reviews. At the same time, they can give comment on other user's review. That's why at first we create a user and they give grant on search view and have the privileges to insert values on review table.

```
mysql> create user 'critiques_user'@'%' IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT select  ON yelp_db.search TO  'critiques_user'@'%';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT  INSERT ON yelp_db.review TO 'critiques_user'@'%';
Query OK, 0 rows affected (0.00 sec)
```

3. Business Analyst can perform data mining and analysis and for that we grant create routine, alter routine and execute privileges. And at the same time, they can create view and show view.

```
mysql> CREATE USER 'businessAnalyst_user'@'localhost' IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT SELECT ON yelp_db.* TO 'businessAnalyst_user'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT CREATE VIEW,SHOW VIEW ON yelp_db.* TO 'businessAnalyst_user'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT CREATE ROUTINE,ALTER ROUTINE,EXECUTE ON yelp_db.* TO 'businessAnalyst_user'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

4. Developers can do cleaning and analysis as well as insert, update, delete. That's why we give following privileges to those users.

```
mysql> CREATE USER 'developer_user'@'localhost' IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT CREATE ON yelp_db.* TO 'developer_user'@'localhost' IDENTIFIED BY 'password';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> GRANT SELECT,DROP,INDEX,REFERENCES,TRIGGER,CREATE ROUTINE,ALTER ROUTINE,EXECUTE,CREATE VIEW,SHOW VIEW, INSERT, UPDATE,DELETE ON yelp_db.* TO 'developer_user'@'l
calhost';
Query OK, 0 rows affected (0.00 sec)
```

5. As the admin can access full database. So we grant ALL to them.

```
mysql> create user 'admin_user'@'localhost' IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT ALL PRIVILEGES ON yelp_db.* TO 'admin_user'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```