

Yelp Project Report

ECE 656

Winter 2018

Ye Wang 20730349 y2886wang@uwaterloo.ca
Yuqing Zhao 20702958 y479zhao@uwaterloo.ca
University of Waterloo

1. Introduction

In this project, we analyze the Yelp dataset to see what knowledge can be gleaned from it. This project can be divided to four parts. First, we construct an ER Diagram to analyze the relationships between entities. Second, we clean the dataset by doing a sanity check. Third, we add an index to improve the efficiency of SQL query running. Then, we analyze the data from three parts: 1. Whether a business is declining or improving in its rating 2. How many stars the majority of users would like to give to the restaurant they have visited? 3. Predict what rating a user will give a business based on how they have rated other businesses and how others have rated that business? In the last part, we create a unique user for each of the specified cases and grant each user the appropriate permissions required to accomplish their task.

2. Before analysis

Three things must be done prior to even starting an analysis:

1. Constructing an E-R diagram
2. Data cleaning
3. Data indexing

2.1 Database Schema Design (E-R Diagram)

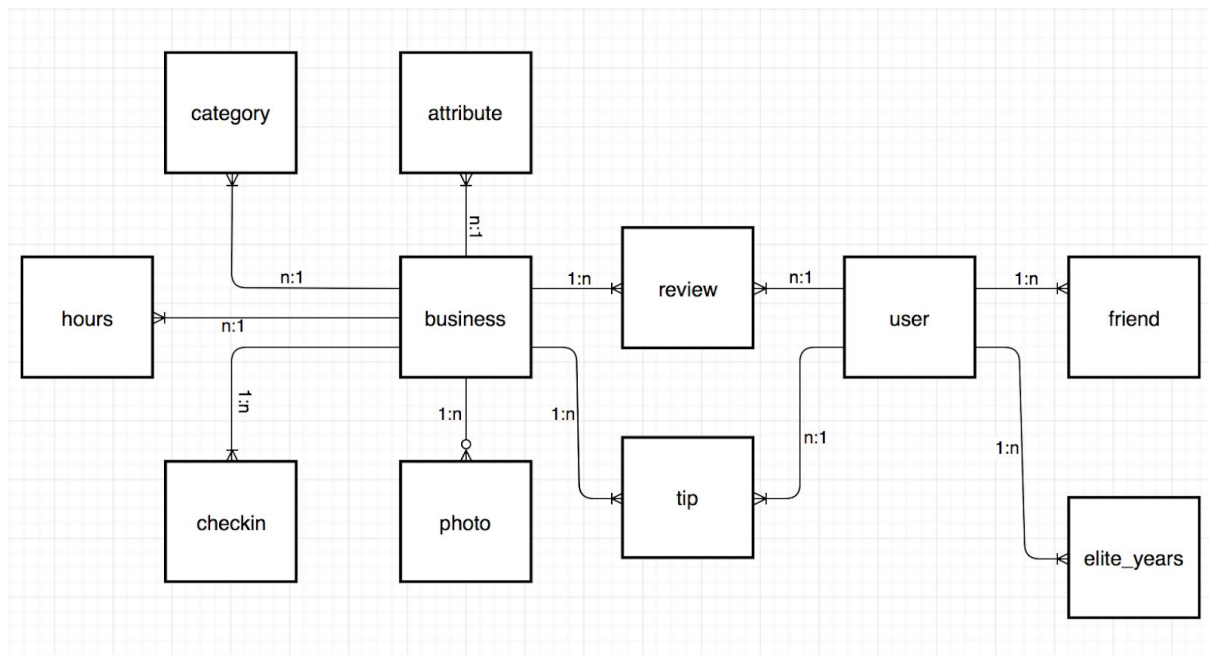


Figure 2.1 E-R Diagram for the Yelp dataset

We designed the E-R diagram (Figure 2.1) for Yelp data. In the database schema, we have 11 entities which are: hours, checkin, category, business, attribute, photo, review, tip, user, friend, and elite_years. Between each entity has different relationships. Given a many-to-one relationship between review and user, we use review table to store the review information such as the specific time of writing reviews that users left. One business can get multiple reviews from users and one review is related to exactly one business. One business can have different tips from users and one tip is given to one business. One person can be an elite user in 1 year or every year. Elite_year table includes some information about when people became elite users. We also create friend table which is related to user table. Checkin table, hours table, attribute table, photo table and category table are related to business table where there are some related information stored within each table.

2.2 Data Cleaning (sanity checks)

For basic sanity checking, data has to be viewed as correct and we need to check both self-consistency and consistency with reality. In this project the basic sanity checking include:

1. The identified Years of Elite must be both consistent with the time that user started reviewing for Yelp and with our own knowledge of current data. As is shown in the following SQL query (Figure 2.2.1).

```
delete elite_years from elite_years inner join
(select e.user_id
 from elite_years e left join user u on e.user_id = u.id where u.id is null) as table1
on elite_years.user_id = table1.user_id;
```

Figure 2.2.1

2. The user cannot exist since Yelp did not start until 2004. As is shown in the following SQL query (Figure 2.2.2).

```
delete from elite_years
where year < 2004 || year > 2018;

delete from review
where date < '2004-09-30' || date > '2018-04-30';

delete from tip
where date < '2004-09-30' || date > '2018-04-30';

delete from user
where yelping_since < '2004-09-30' || yelping_since > '2018-04-30';
```

Figure 2.2.2

3. The user_id in review table should exist in both business table and user table. As is shown in the following SQL query (Figure 2.2.3).

```
delete review from review inner join
(select review.business_id
 from review left join business on review.business_id = business.id where business.id is null) as table1
on review.business_id = table1.business_id;

delete review from review inner join
(select review.user_id
 from review left join user on review.user_id = user.id where user.id is null) as table1
on review.user_id = table1.user_id;
```

Figure 2.2.3

4. The number of reviews written by a user as recorded in the user table cannot be smaller than the number of reviews in the business table. As is shown in the following SQL query (Figure 2.2.4).

```
delete user from user inner join
(select tmp.user_id from user u inner join
(select user_id, count(*) total from review group by user_id) as tmp
on u.id = tmp.user_id where tmp.total > u.review_count) as table1
on user.id = table1.user_id;
```

Figure 2.2.4

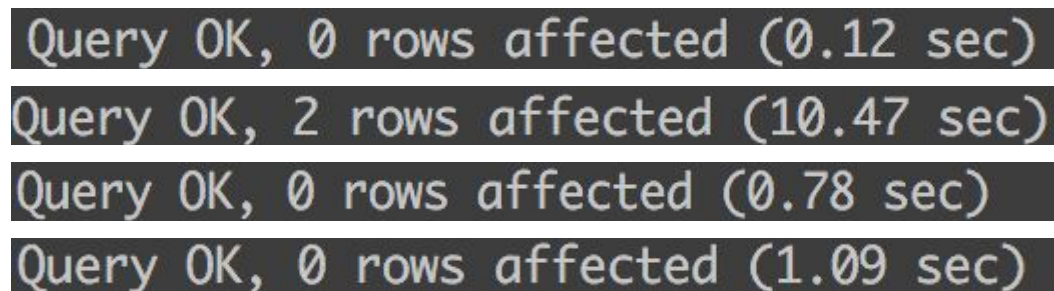
2.3 Indexing

In this portion, we determine what indexes are necessary to enable efficient querying in both the cleaning and analysis parts.

1. For “without index” portion of cleaning, we remove all primary and foreign keys from our database. The result of cleaning is shown as follows:

For cleaning case 1, 3, 4 above: the time to execute queries exceeds a reasonable thresholds(because of a sizable joins). We terminate the related queries after implementing one hour.

For cleaning case 2 above, the result is shown as follows (Figure 2.3.1):



```
Query OK, 0 rows affected (0.12 sec)
Query OK, 2 rows affected (10.47 sec)
Query OK, 0 rows affected (0.78 sec)
Query OK, 0 rows affected (1.09 sec)
```

Figure 2.3.1

2. After adding primary keys, foreign keys and index of cleaning, the result is shown as follows:

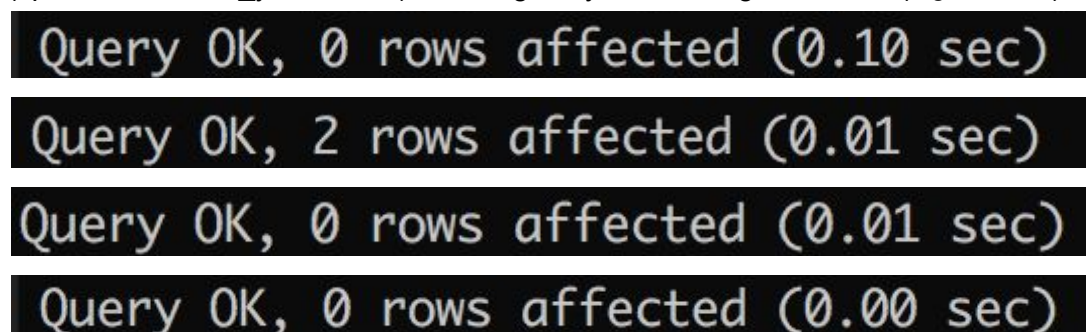
For cleaning case 1 above, we add user_id (elite_years table) as B+-tree index, id (elite_years table) as primary key and user_id (elite_years table) as foreign key referencing business table (Figure 2.3.2).



```
Query OK, 6 rows affected (0.77 sec)
```

Figure 2.3.2

For cleaning case 2 above, we add year (elite_years table), date (review table), yelping_since (user table) as B+-tree index, id (elite_years table, user table, tip table, review table) as primary key, user_id (review table) as foreign key referencing user table, user_id (tip table and elite_years table) as foreign key referencing user table (Figure 2.3.3).



```
Query OK, 0 rows affected (0.10 sec)
Query OK, 2 rows affected (0.01 sec)
Query OK, 0 rows affected (0.01 sec)
Query OK, 0 rows affected (0.00 sec)
```

Figure 2.3.3

For cleaning case 3 above, we add business_id (review table) and user_id (review table) as B+-tree index, id (review table) as primary key, user_id (review table) as foreign key referencing user table and business_id (review table) as foreign key referencing business table (Figure 2.3.4).

Query OK, 0 rows affected (42.74 sec)

Query OK, 7528 rows affected (2 min 52.85 sec)

Figure 2.3.4

For cleaning case 4 above, we add user_id (review table) and review_count (user table) as B+-tree index, id (user table) as primary key, id (review table) as primary key and user_id (review table) as foreign key referencing user table (Figure 2.3.5).

Query OK, 1323 rows affected (4 min 5.85 sec)

Figure 2.3.5

3. For “without index” portion of analyzing, we remove all primary and foreign keys from our database. The result of cleaning is shown as follows (Figure 2.3.6):

296 rows in set (15.68 sec)

avg_star	user_count
0-1	87670
1-2	54282
2-3	181239
3-4	444758
4-5	556829

5 rows in set (0.95 sec)

Figure 2.3.6

4. After adding primary keys, foreign keys and index of analyzing, the result is shown as follows (Figure 2.3.7):

For data case analyze, to improve the performance of query execution, we add business_id of review as a foreign key referencing to business. We add average_stars of user as index to boost the performance of query execution (Figure 2.3.7).

296 rows in set (0.20 sec)

avg_star	user_count
0-1	87670
1-2	54282
2-3	181239
3-4	444758
4-5	556829

5 rows in set (0.39 sec)

Figure 2.3.7

3. Data Analysis

After cleaning the dataset, we use some queries to get useful data and find out the important information behind it. There are three things that we analyse. First, we analyse whether a business is declining or improving in its rating. Second, we analyse the fraction of user counts per average star. Third, we analyse and predict what rating a user will give a business based on how they have rated other businesses and how others have rated that business.

3.1 Whether a business is declining or improving in its rating?

In that part, we use Python application to query the database in order to plot the rating trend graph of a certain business. At total, We select three typical businesses to analyse, PIO PIO, SUNNYSIDE GRILL and Dial Carpet Cleaning.

Recent rating could hardly influence average rating dramatically due to a big amount of data before. However, it is very important for a business to notice the change in current state.

As figure 3.1 shown, we select date and count the average stars grouped by date, in terms of a certain business.

```
select date, avg(stars) as avg_stars
from review
where business_id = '--cZ6Hhc9F7VvkKXxHMOVZSQ' group by date order by date ;
```

Figure 3.1 The SQL Query to Select Date and Avg_stars

This query is executed by python supported by mysql connector module. As the figure 3.2 shown, after a slight rise, the rating of PIO PIO goes down modestly and below 4 stars. As the figure 3.3 shown, there is a rise in recent rating of SUNNYSIDE GRILL. As the figure 3.4, the average rating of SDIAL CARPET CLEANING remains approximately unchanged in the recent period.

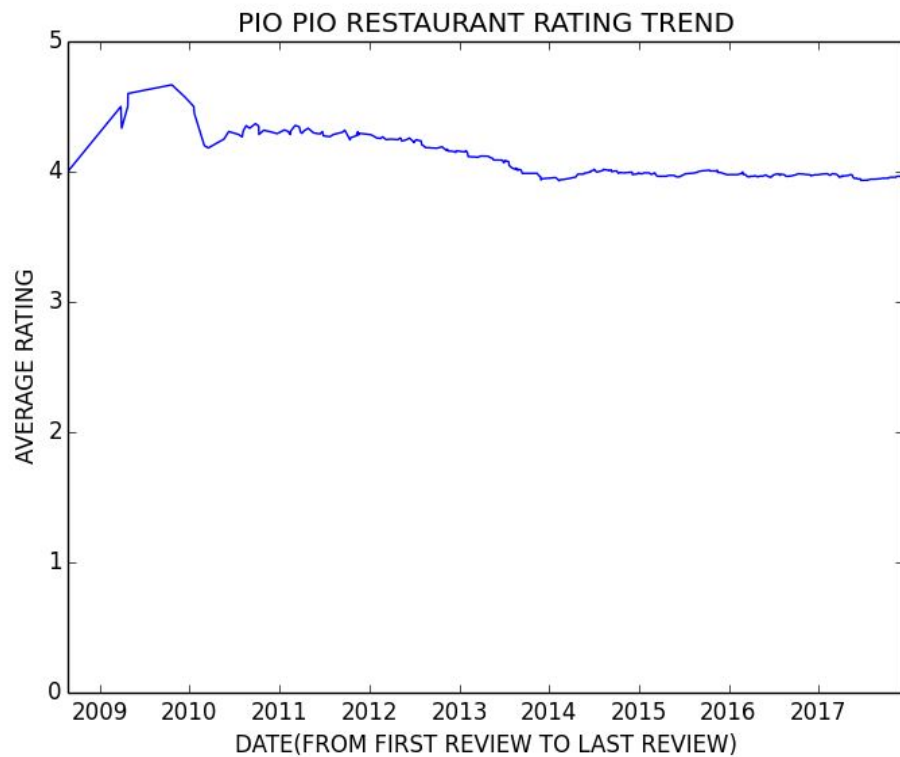


Figure 3.3 The Change of Average rating of PIO PIO along Time

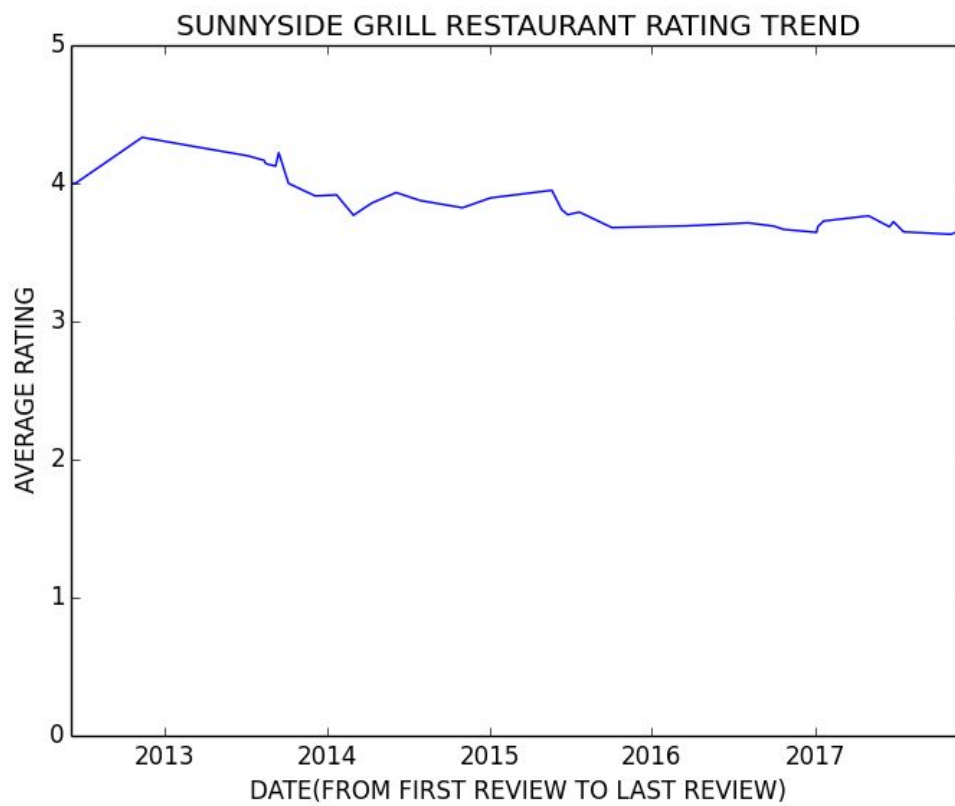


Figure 3.4 The Change of Average Rating of Sunnyside Grill along Time

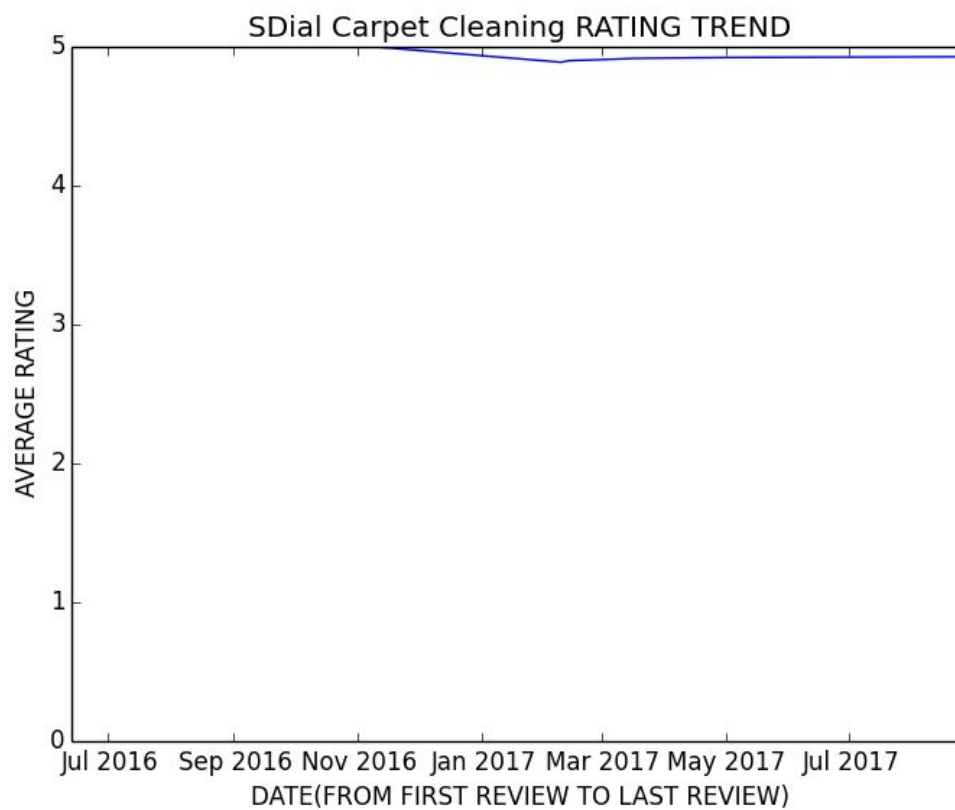


Figure 3.5 The Change of Average Rating of SDial Carpet Cleaning along Time

3.2 How many stars that the majority of users would like to give to the restaurant they have visited?

In this part, we are interested in overall generosity of users and their rating styles. firstly, we create a view to store the number of users per average_stars, as shown in figure 3.6. Then, we separate those rating data into 5 ranges and count the number of users for each bin, as shown in figure 3.7.

Here we create a generate_series_base procedure which could help with data analyse in different bins.

In conclusion, most of users are willing to give 4-5 stars to restaurants they have experienced which is the highest rating level.

```
• drop view if exists avgstar_usernum;

• create view avgstar_usernum as
select average_stars, count(id) as usernum
from user
group by average_stars
order by average_stars desc;

• DROP PROCEDURE IF EXISTS generate_series_base;
DELIMITER $$
• CREATE PROCEDURE generate_series_base (IN n_first BIGINT, IN n_last BIGINT)
BEGIN
    -- Call generate_series_n_base stored procedure with "1" as "n_increment".
    CALL generate_series_n_base(n_first, n_last, 1);
END $$
DELIMITER ;

• DROP PROCEDURE IF EXISTS generate_series_n_base;
DELIMITER $$
• CREATE PROCEDURE generate_series_n_base (IN n_first BIGINT, IN n_last BIGINT, IN n_increment BIGINT)
BEGIN
    -- Create tmp table
    drop table if exists series_tmp;
    CREATE TABLE series_tmp (
        series bigint
    ) engine = memory;

    WHILE n_first <= n_last DO
        -- Insert in tmp table
        INSERT INTO series_tmp (series) VALUES (n_first);

        -- Increment value by one
        SET n_first = n_first + n_increment;
    END WHILE;
END $$
DELIMITER ;

• call generate_series_base(1,5);

• select concat(g.series-1, '-', g.series) as avg_star, sum(usernum) as user_count
from avgstar_usernum as a, series_tmp as g
where a.average_stars <= g.series and a.average_stars > g.series-1
group by g.series
order by g.series;
```

Figure 3.6 SQL queries to get the number of uses for each bin of user average_stars

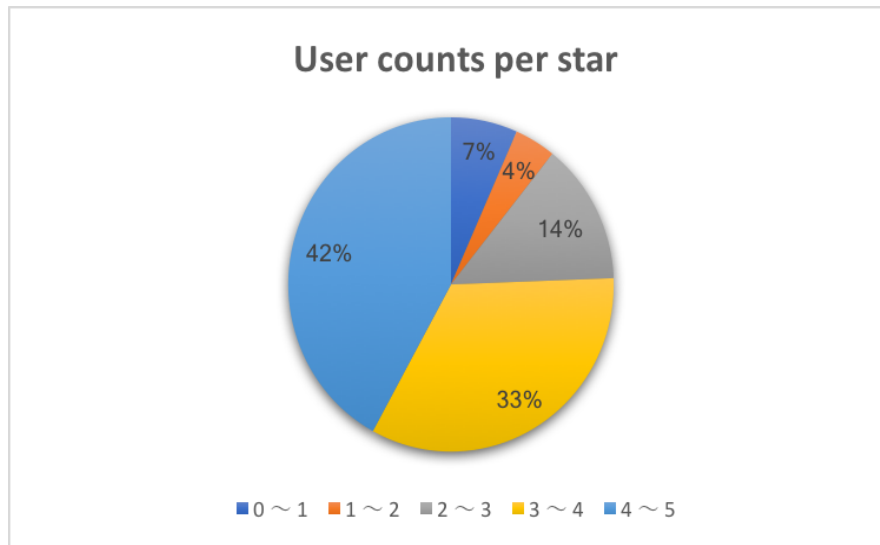


Figure 3.7 Pie chart of user counts for each average_stars bin

3.3 Predict what rating a user will give a business based on how s/he has rated other businesses and how others have rated that business?

We use K-Nearest-Neighbors (KNN), a powerful classification algorithm to analyze this question. we implement this algorithm in Python.

At the first step, we select 500 business-user pairs (500 rows), queries as shown in figure 3.8 and partial data (10 rows) as shown in figure 3.9 . There are two features. One is the average star that other users rate this business. The whole average stars would not be influenced too much by one user so we use stars attribute of table business to represent it. The other one is the average star this user rate other businesses. The whole average stars would not be influenced too much by one user so we use stars attribute of table business to represent it. The label of this model is what star this user rate for this business. From the training model, we can predict how a user rate for a business.

Based on our model, we found users who give higher rates to other businesses normally are willing to give higher rates to a certain business. the comparison of two users are shown in figure 3.10 and 3.11. This prediction result is reasonable.

In our validation, we check whether this model is reasonable. Given 500 data pairs, we know the actual values of thisUser_rate_thisBusiness and we compare those with predicate results. we calculate the error percentage of our model. The error percentage is fail pairs / total pairs (500). The final result is 0.152 based on the 500 rows selected from the datasat as shown in figure 3.12. We could improve the performance of this training model by increasing training sample size.

```
select business_id, user_id as user_id, business.stars as otherUser_rate_thisBusiness, user.average_stars as thisUser_rate_otherBusiness, review.stars as thisUser_rate_thisBusiness
from business inner join review on business.id = review.business_id inner join user on user.id = review.user_id
order by business_id limit 500;
```

Figure 3.8 queries to select user-business pairs as training data

business_id	user_id	otherUser_rate_thisBusiness	thisUser_rate_otherBusiness	thisUser_rate_thisBusiness
--6MefnULPED_I942VcFNA	Aj0jZyvbqolaSa5BRaYcHA	3	2.47	1
--7zmmkVg-IMGaXbuVd0SQ	D1Xarj0DURiDASeXck39yQ	4	5	5
--8LPVSo5i0Oo61X01sV9A	stvMEKrdmvmwChrvEkXqyRw	4.5	4.36	5
--9e1ONYQuAa-CB_Rrw7Tw	GQWk8vgYGIN9hp0XP0V05w	4	3.86	5
--9QQLMTbFzLJ_oT-ON3Xw	NS2OQzrmJYHRXboibOYFDA	3.5	3.04	5
--ab39IjZR_xUf81WyTyHg	jKDTqRBKpp3tjibJHDoLRg	4	4.22	5
--cjBEbXMI2obtaRHNSFrA	eBs3CiWDg0Mk5h8tvqdYXg	3	3.5	3
--cZ6Hhc9F7VkkKXxHmVZSQ	UwzguhtWCmffqjJ9rf6NOQ	4	4.08	5
--SrzipvFLwP_YFwB_Cetow	isJEo2IISFonbenri9KQcw	3.5	3.17	3
--ttCFj_csKJhnaMRNuiw	GS8-hNHu_CLEMGj993TCA	3.5	4.67	5

Figure 3.9 partial training data

```
Average_stars of this business: 4
Average_stars given by this user: 3.5
Predict how many stars this user will give to this business: 4
```

Figure 3.10 predict users who are willing to give higher rates with 3.5 average stars

```
Average_stars of this business: 4
Average_stars given by this user: 2
Predict how many stars this user will give to this business: 2
```

Figure 3.11 predict users who are willing to give lower rates with 2 average stars

```
validation pass
error percentage:0.152
```

Figure 3.12 the error percentage of this classification model

4. Applications for Different Users

In this part, we create several types of users which grant them different permission to access the Yelp database.

4.1 Logged In User

A logged in user should only be provided enough privileges to write the review. As is show in the following SQL (Figure 4.1).

```
# making a logged in user
CREATE USER 'Log_in_user'@'localhost' IDENTIFIED BY 'password';
GRANT INSERT ON yelp_db.review TO 'Log_in_user'@'localhost';
FLUSH PRIVILEGES;
```

Figure 4.1

4.2 Business Analysts

Business analysts cannot perform IUD (Insert/Update/Delete) operations on the database but should have access to creating extra views and show views on the database schema. As is show in the following SQL (Figure 4.2).

```
# making a Business analysts user
CREATE USER 'Analysts'@'localhost' IDENTIFIED BY 'password';
GRANT CREATE VIEW ON yelp_db . * TO 'Analysts'@'localhost';
GRANT SHOW VIEW ON yelp_db . * TO 'Analysts'@'localhost';
FLUSH PRIVILEGES;
```

Figure 4.2

4.3 Developers

Developers working with this database are able to create new tables and perform data cleaning and indexing. They are allowed to perform IUD operations on the database. As is show in the following SQL (Figure 4.3).

```
# making a Developers user
CREATE USER 'Developers'@'localhost' IDENTIFIED BY 'password';
GRANT CREATE ON yelp_db . * TO 'Developers'@'localhost';
GRANT INDEX ON yelp_db . * TO 'Developers'@'localhost';
GRANT INSERT ON yelp_db . * TO 'Developers'@'localhost';
GRANT UPDATE ON yelp_db . * TO 'Developers'@'localhost';
GRANT DELETE ON yelp_db . * TO 'Developers'@'localhost';
FLUSH PRIVILEGES;
```

Figure 4.3

4.4 Database Admin

Database administrator who has full access over the database. As is show in the following SQL (Figure 4.4).

```
# making a Database_Admin user
CREATE USER 'Admin'@'localhost' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON yelp_db . * TO 'Admin'@'localhost';
FLUSH PRIVILEGES;
```

Figure 4.4

5. Conclusion

In this project, we analyzed the yelp dataset and find out something useful.

For data cleaning, facing the complicated queries such as sizable joins, we need to add index which can be more effective and faster to execute queries.

For designing ER diagram and database schema, we need to predict what data analysis and procedures will be used to make it suitable for our future needs, and also need to consider that we have to keep integrity of the original data when loading them into our database.

For data analysis, we focus on the rating information of those tables which is the most useful part of the dataset. Use python to analyse the SQL results and plot the results to get more direct relationships between different kinds of data.

6. References

1. <https://www.digitalocean.com/community/tutorials/how-to-create-a-new-user-and-grant-permissions-in-mysql>

2. https://dev.mysql.com/doc/refman/5.7/en/privileges-provided.html#priv_update