ECE 656 Project Report: Data-Bites

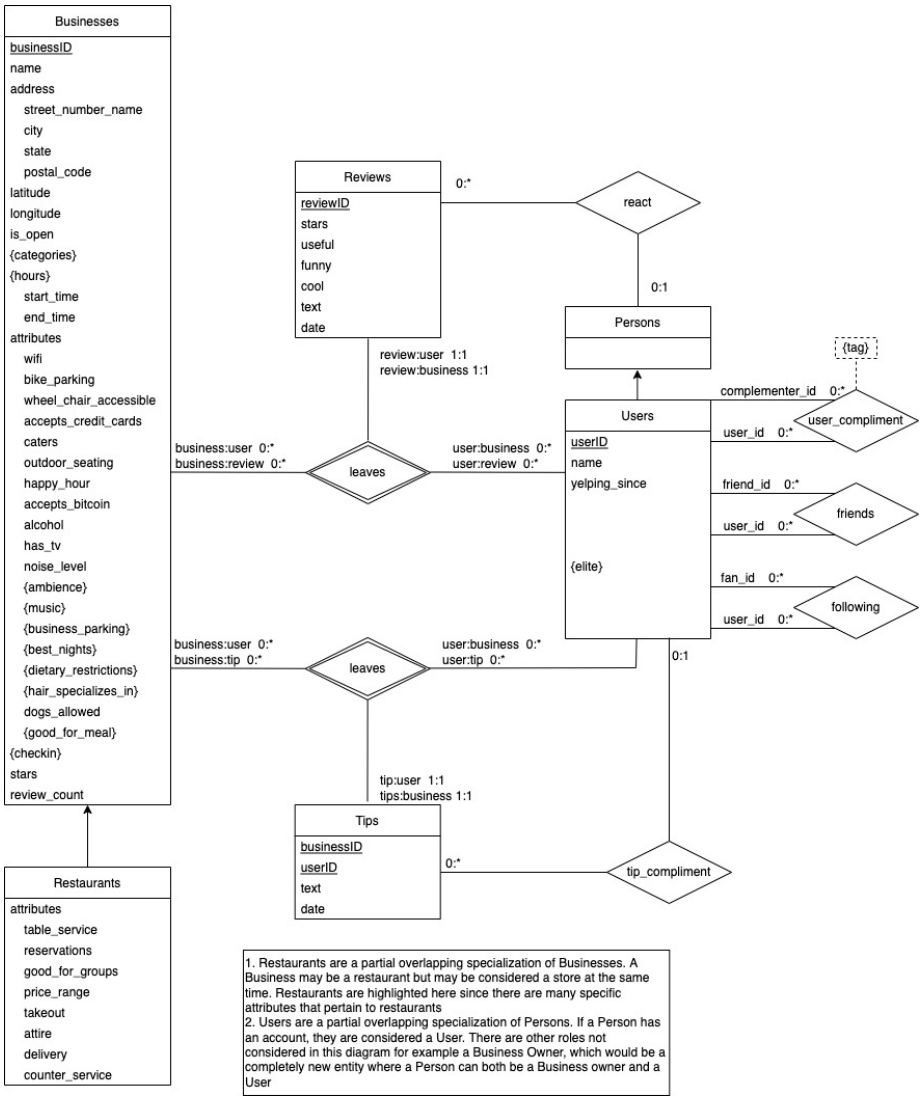
Connie Ho
c72ho@uwaterloo.ca
20955691

Andy Gehlan
agehlan@uwaterloo.ca
20474701

Data Bites is an application modelled after the Yelp dataset that allows people to discover local businesses such as restaurants or stores. The application is intended to be a crowd-sourced business review and social networking site. Users can submit review of their services on a fixed rating scale and businesses can provide listing information and other details about themselves. Additionally, users may react to reviews or interact with other users. This report will discuss the ER Model, the relational schema, client/server application, and data mining. Discussions of design choices and tradeoffs will be within the ER Model and relational schema sections.

ER Model

The following diagram is a snapshot of the Data Bites ER Model:



The entities present in the ER model are Businesses, Restaurants, People, Users, Reviews, Tips. Persons in this setting is anyone with a computer that can interact with the site in general, there are no attributes in this entity as they remain completely anonymous. A User is a partial, overlapping specialization of a Person with the attributes user_id, name, yelping_since, and the multivalued attribute elite (years in which they were of elite status). There exists other Persons that are for example Businesses Owners that can also be Users at the same time but are not modelled here as they were not given in the dataset.

Restaurants are also a partial, overlapping specialization of Businesses. They contain a set of attributes specific to Restaurants such as table_service, reservations, good_for_groups, price_range, takeout, attire, delivery, and counter_service. Businesses can be many specializations. For example, a Business can be considered a Restaurant or Store at the same time. Other specializations are not shown in the ER model as the dataset does not have clear separation of attributes for anything other than a Restaurant.

Businesses are in a ternary relationship with Users and Reviews. In other words, a User leaves a Review on a Business. Reviews are modelled as a separate entity rather than a relationship set because of the complexity of the attributes which include the business_id, user_id, stars, text, date and tags such as useful, funny, and cool. A Review is then considered as a weak entity because it cannot exist without Businesses or Users. The candidate key of the entity set is (business_id, user_id). However, to simplify the entity set, a unique review_id is given to each review and is chosen to be the primary key. In the future, if users were given the functionality to add more than one review for a business, the review_id could continue to be used as the primary key and allows for better scalability. The attributes (business_id and user_id) are given a unique constraint when translating to the relational model. The cardinalities are as follows:

- Business:user (0:*) a business may have 0 or more users reviewing it
- Business:review (0:*) a business may have 0 or more reviews
- Review:user (1:1) - a review must have exactly 1 user writing it
- Review:business(1:1) - a review must be exactly for one business
- User:review (0:*) - a user may write 0 or many reviews
- User:business (0:*) - a user may leave reviews for 0 or more businesses

Tips are similar to Reviews. Tips are also modelled in a ternary relationship with Businesses and Users – A User leaves Tips on a Business. Tips are also modelled as a weak entity rather than a relationship given that there are complex attributes. One could argue Reviews are a specialization of Tips however there are important differences and therefore the two entities are modelled as separate. Firstly, for a given Business, a User is allowed to leave at most 1 Review but is allowed to leave many Tips. The primary key for the Tips entity must include an additional attribute and is chosen to be (business_id, user_id, date). An alternative that would have been preferred is to have a unique tip_id to simplify the primary key, however this was not given in the dataset. Another difference is that Tips are intended to be short comments. Although both have a text attribute, the maximum length the text of a Tip can be is 500 characters while a Review can have 5000. Interaction with Tips and Reviews are also different. Any Person may react to a Review by tagging it ‘useful’, ‘funny’ or ‘cool’. For a Tip, a Person must be a User and

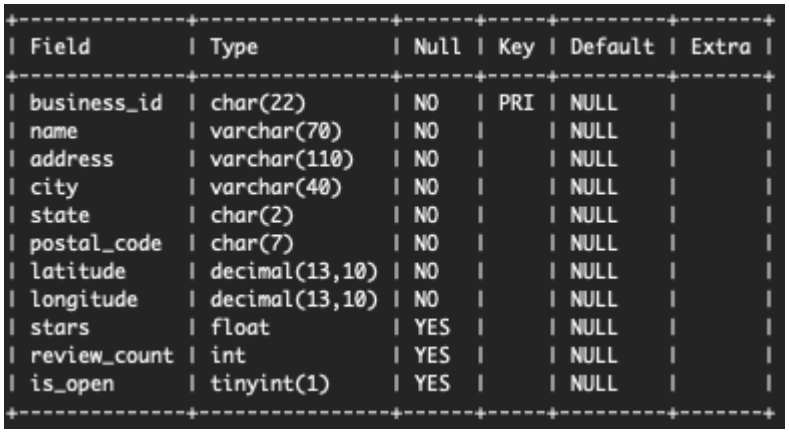
can only leave compliments. Since only compliment counts are given in the Yelp dataset, a compliment in this context, is acknowledging another User's Tip for being good.

The User entity has many roles in the ER model. First, a User can have many friends in which a friend must also be a User. Both Users involved in the relationship must be friends with each other. Secondly, Users can also have many followers and follow other Users. In this case, for example, User1 may follow User2 however User2 does not have to follow back User1. Users may also interact with other Users through compliments. This is similar to compliments for Tips. Since only a complement count is given for a User, in the context of this project, a user compliment is merely a user-to-user interaction where one user acknowledges another User (similar to liking another User).

Relational Model

Note the relational model can be found in the scripts folder and should be ran in order of 1_create_tables.sql, 2_load_data.sql, 3_load_keys.sql. then 4_add_indexes. The create tables file contain the table creations along with primary keys, unique, null and default constraints specified. The 3_load_keys file contains all foreign keys, other constraints and altering table commands. The 4_add_indexes file contain the relevant indexes to improve performance on the client application. Data must be loaded within the scripts folder.

The Business entity when translated to a relational schema is represented by the following table:



Field	Type	Null	Key	Default	Extra
business_id	char(22)	NO	PRI	NULL	
name	varchar(70)	NO		NULL	
address	varchar(110)	NO		NULL	
city	varchar(40)	NO		NULL	
state	char(2)	NO		NULL	
postal_code	char(7)	NO		NULL	
latitude	decimal(13,10)	NO		NULL	
longitude	decimal(13,10)	NO		NULL	
stars	float	YES		NULL	
review_count	int	YES		NULL	
is_open	tinyint(1)	YES		NULL	

The primary key is the business_id. The address column represents the street name and street number and was kept separate from city and state as these would help with searching on the client application. By doing this, an index was added on city and state to improve query performance. Although review_count and stars (the average rating) are not atomic attributes, the decision was made to keep these as attributes in the Businesses relation. This is to primarily increase performance at the cost of highly accurate data. For example, if left as a computation by joining the Reviews table, if a user wants to know what businesses 3 stars are over, the query will have to be executed over millions of businesses and several thousands of reviews potentially leading to long search times. In 2020-2021 over 400K new businesses were added to the

database. The trade-off for efficiency is accuracy of the stars and review count. A few not up-to-date reviews in 1000s of reviews for a business will not make a large difference in the everyday user experience.

Business attributes were translated to an 'Attributes' table with each business attribute existing as a possible domain value instead of as a column to deal with large quantities of missing information. The primary key is therefore (business_id, attribute) with business_id as a foreign key referencing Businesses(business_id). For example, the attribute ''accepts_bitcoin" is a Boolean value with most businesses having NULL or false values. With the business attribute as a domain value instead, having an entry in the attributes table simply means it is True. The tradeoff is that the User no longer knows whether the attribute is false or simply unknown. However, since there are thousands of categories, they likely do not want to know the status of all thousand at once. Business Categories were constructed in a similar manner to Attributes. The primary key is (business_id, category) with business_id as a foreign key referencing Businesses(business_id). An index was also added on categories as this would be a common value Users would be able to filter businesses on. Ideally, attributes would also have an index, but the functionality was not built in this iteration of the application. The enum datatype was considered for attributes and categories however there are dozens of attributes and almost 1000 categories and therefore the varchar datatype was chosen to allow more flexibility in the application (especially as more attributes and categories are added as the applications scale). The following illustrates an example query on the Attributes table for a particular business:

```
mysql> select * from Attributes WHERE business_id="ZZoh62g4ivFlaeBHk9hJOg";
+----------------------+---------------------------+
| business_id          | attribute                 |
+----------------------+---------------------------+
| ZZoh62g4ivFlaeBHk9hJOg | BusinessAcceptsBitcoin    |
| ZZoh62g4ivFlaeBHk9hJOg | BusinessAcceptsCreditCards |
+----------------------+---------------------------+
2 rows in set (0.00 sec)
```

Multivalued business attributes such as Music were translated into their own tables. The primary key is also (business_id, attribute) with business_id as a foreign key referencing Businesses(business_id). This was done to prevent duplication of data in the Attributes table. However, by separating out the multivalued attributes into their own tables, the server application became very clunky as shown in the following code:

```
query = """
SELECT Attributes.attribute, Ambience.attribute, Best_Nights.attribute, Business_Parking.attribute,
Dietary_Restrictions.attribute, Good_For_Meals.attribute, Hair_Specializes_In.attribute, Music.attribute FROM Attributes
LEFT JOIN Ambience USING(business_id)
LEFT JOIN Best_Nights USING(business_id)
LEFT JOIN Business_Parking USING(business_id)
LEFT JOIN Dietary_Restrictions USING(business_id)
LEFT JOIN Good_For_Meals USING(business_id)
LEFT JOIN Hair_Specializes_In USING(business_id)
LEFT JOIN Music USING(business_id)
WHERE business_id=%s"""
```

If another multivalued attribute were created, this would have to be joined to Attributes table to get all the business attributes. A view should be considered for future applications to simplify this query.

Business hours were made into its own table with each day having an opening and closing column. Previously each day was a single string of opening and closing times (e.g. 'Monday: 12 PM - 6 PM'). Providing the opening and closing times for each date gave more flexibility when working with the data.

Field	Type	Null	Key	Default	Extra
business_id	char(22)	NO	PRI	NULL	
Monday_Open	char(5)	YES		NULL	
Monday_Close	char(5)	YES		NULL	
Tuesday_Open	char(5)	YES		NULL	
Tuesday_Close	char(5)	YES		NULL	
Wednesday_Open	char(5)	YES		NULL	
Wednesday_Close	char(5)	YES		NULL	
Thursday_Open	char(5)	YES		NULL	
Thursday_Close	char(5)	YES		NULL	
Friday_Open	char(5)	YES		NULL	
Friday_Close	char(5)	YES		NULL	
Saturday_Open	char(5)	YES		NULL	
Saturday_Close	char(5)	YES		NULL	
Sunday_Open	char(5)	YES		NULL	
Sunday_Close	char(5)	YES		NULL	

The Users relational schema is represented by the following table:

Field	Type	Null	Key	Default	Extra
user_id	char(22)	NO	PRI	NULL	
name	varchar(35)	YES		NULL	
yelping_since	datetime	YES		NULL	
compliment_hot	int	YES		0	
compliment_more	int	YES		0	
compliment_profile	int	YES		0	
compliment_cute	int	YES		0	
compliment_list	int	YES		0	
compliment_note	int	YES		0	
compliment_plain	int	YES		0	
compliment_cool	int	YES		0	
compliment_funny	int	YES		0	
compliment_writer	int	YES		0	
compliment_photos	int	YES		0	

The average_stars, review_count, and counts for review tags (useful, funny, cool) were dropped from the initial dataset to reduce error, and if needed could be obtained by joining the Reviews table using the business id. User details are not likely to be frequently searched upon, and there are far fewer reviews written by a user than for a business. Compliment counts are left in the user table to be used for potential data mining/analysis. A compliment is typically between two users; however, this data is not provided in the dataset (possibly for anonymity reasons).

The Reviews relational schema is represented by the following table:

Field	Type	Null	Key	Default	Extra
review_id	char(22)	NO	PRI	NULL	
user_id	char(22)	NO	MUL	NULL	
business_id	char(22)	NO	MUL	NULL	
stars	int	YES		0	
useful	int	YES		0	
funny	int	YES		0	
cool	int	YES		0	
text	text	YES		NULL	
date	datetime	YES		NULL	

A User or Person can interact with a Review via the tags ‘useful’, ‘funny’ and ‘cool’. An alternative considered was having a guest user where each time an anonymous user liked a review, an entry would be added to a Reviews_Reactions table with either NULL or a guest user_id. This could result in a very large table considering the number of Reviews that currently exist and would need to have many other details such as the user_id (or lack thereof), date or cumulative count to create the appropriate primary key. The application in this project is rather simple and does not need to look at historical count data and therefore these tag counts are kept in the Reviews schema as columns.

The Tips relational schema is represented by the following table:

Field	Type	Null	Key	Default	Extra
user_id	char(22)	NO	PRI	NULL	
business_id	char(22)	NO	PRI	NULL	
text	text	YES		NULL	
date	datetime	NO	PRI	1970-01-01 00:00:00	
compliment_count	int	YES		0	

The primary key is (user_id, business_id, date) as a User can leave multiple tips on a Business. A Tip_Compliment table was created to demonstrate the relationship between Users and Tips although complimenter_id is not provided in the initial dataset. The attribute compliment_count is therefore left for potential data-mining exercises later but will not be interactable in the client application. Since there is some data with null values for date, and date is part of the primary key, the arbitrary UTC start date was chosen to be the default value.

Server & Client Application

Flask was used as the framework to build the server/client application. The server connects to the database using the flask-mysql library. The client application is produced in the form of html

templates. Test cases for the server and client application can be found in the app/tests founder in pseudo-code. Features implemented in the client application include:

1. Creating a user. The user_id is the first letter of the first name, last name up to 10 digits and the remaining digits up to 22 being random numbers. The ideal client application would include more security features such as password and logging in and out to establish user permissions.
2. Viewing user details such as when they first started yelping, the average rating they've given the number of reviews, compliments, and friends they have.
3. Adding a user as a friend. The ideal client application would have a friend request system as a friend and a user must have a bi-directional relationship. We assume in this application that if a user adds a friend, the friend automatically accepts, and a record is made in the User_Friends table.
4. Searching for businesses. A business can be searched for by name, location (city and state), category and stars (average rating). The ideal client application could also filter by attributes after obtaining the initial results.
5. Viewing business details such as stars, the number of reviews, check ins, hours, categories, attributes, tips and reviews.
6. Adding a review. The ideal client application would implement a login/logout feature where only logged in users can leave reviews.
7. Adding tips. Again, ideally only logged in users should be able to leave tips.
8. Tagging reviews with useful, funny, cool. The ideal client application would keep track of which person reacted to a review either via a guest id or their user id.
9. Checking in. The ideal client application would verify the location of the person with the latitude and longitude.

For future work, the ideal application would include functionality from the business perspective, for example adding or managing a business listing. It would also include more social aspects such as following other users and creating lists for example for favorited businesses.

Data Mining

The yelp database has attributes associated with each business. These attributes seem to be self-reported and are optional. One question we pose is are there any insights from the associations between top and poor rated restaurants? This can be helpful when considering how poorly rated restaurants can improve their ratings. We also ask if there are certain geographical regions that have skewed distributions of their average ratings. This can then lead to further questions on why those regions are different than others.

Association Rules

The first step was to gather all the data needed. In our schema we had to join all the tables that represented attributes which were split because some were multi-valued and others Boolean.

```
-- Getting business ids with their respective attributes
SELECT business_id, stars, attribute FROM Businesses INNER JOIN Attributes USING
(business_id)
UNION ALL
SELECT business_id, stars, attribute FROM Businesses INNER JOIN Ambience USING
(business_id)
UNION ALL
SELECT business_id, stars, attribute FROM Businesses INNER JOIN Best_Nights USING
(business_id)
UNION ALL
SELECT business_id, stars, attribute FROM Businesses INNER JOIN Business_Parking USING
(business_id)
UNION ALL
SELECT business_id, stars, attribute FROM Businesses INNER JOIN Dietary_Restrictions
USING (business_id)
UNION ALL
SELECT business_id, stars, attribute FROM Businesses INNER JOIN Good_For_Meal USING
(business_id)
UNION ALL
SELECT business_id, stars, attribute FROM Businesses INNER JOIN Hair_Specializes_In
USING (business_id)
UNION ALL
SELECT business_id, stars, attribute FROM Businesses INNER JOIN Music USING
(business_id)
INTO OUTFILE '/home/andy/Documents/data-
bites/database/mysql/business_attributes_stars.csv'
FIELDS ENCLOSED BY '"'
TERMINATED BY ';'
ESCAPED BY '"'
LINES TERMINATED BY '\r\n';
```

Above is the query that returns a table with 3 columns: business_id, stars, attribute. This data is then further processed in python. Due to the memory constraints imposed by the algorithms used for determining association rules, the data had to be significantly reduced.

The following filters were applied prior to running the association algorithm:

- Top rated restaurants are those with average stars rating as 4.5 or above
- Poorly rated restaurants are those with average stars rating as 3 or below
- A restaurant must have at least 15 attributes (of 96 total options)
- An attribute must be in at least 2000 businesses (of 140k businesses)

The following filters were applied on the association algorithm:

- 50% support
- 90% confidence

The algorithms apriori and association rules came from the python library mlxtend.frequent_patterns. These algorithms require a one-hot-encoded pandas DataFrame as an input.

With the filters applied there still is a prohibitively large number of associations to manually analyze; thus, we chose to only manually analyze the attributes associated with 'RestaurantsGoodForGroups' as this is a common customer segment restaurants strive for.

Among the top-rated restaurants there were numerous associations involving wheelchair accessibility and the following set of free Wi-Fi, good for kids, casual attire and average noise-level attributes being associated with restaurants good for groups.

These wheelchair attributes were not present in the poorly rated restaurants associations. Upon further inspection, those restaurants that have wheelchair accessibility available tended to perform better than the average restaurant as well.

Among the poorly rated restaurants the common theme seemed to be having a TV among other attributes implying that TVs typically will lead to lower ratings, particularly those restaurants that are targeting groups.

antecedents	consequents
WheelchairAccessible,average NoiseLevel,free WiFi	RestaurantsGoodForGroups
WheelchairAccessible,average NoiseLevel,free WiFi,RestaurantsTakeOut	RestaurantsGoodForGroups
casual RestaurantsAttire,average NoiseLevel,free WiFi,WheelchairAccessible	RestaurantsGoodForGroups
WheelchairAccessible,average NoiseLevel,BusinessAcceptsCreditCards,free WiFi	RestaurantsGoodForGroups
average NoiseLevel,casual RestaurantsAttire,free WiFi,RestaurantsTakeOut,WheelchairAccessible	RestaurantsGoodForGroups
average NoiseLevel,free WiFi,RestaurantsTakeOut,WheelchairAccessible,BusinessAcceptsCreditCards	RestaurantsGoodForGroups
WheelchairAccessible,average NoiseLevel,free WiFi,GoodForKids	RestaurantsGoodForGroups
average NoiseLevel,casual RestaurantsAttire,free WiFi,WheelchairAccessible,BusinessAcceptsCred…	RestaurantsGoodForGroups
average NoiseLevel,free WiFi,GoodForKids,RestaurantsTakeOut,WheelchairAccessible	RestaurantsGoodForGroups
average NoiseLevel,casual RestaurantsAttire,free WiFi,GoodForKids,WheelchairAccessible	RestaurantsGoodForGroups
average NoiseLevel,casual RestaurantsAttire,free WiFi,RestaurantsTakeOut,WheelchairAccessible,…	RestaurantsGoodForGroups
WheelchairAccessible,free WiFi,BikeParking	RestaurantsGoodForGroups
WheelchairAccessible,BikeParking,casual,free WiFi	RestaurantsGoodForGroups
WheelchairAccessible,BikeParking,free WiFi,RestaurantsTakeOut	RestaurantsGoodForGroups
casual RestaurantsAttire,BikeParking,free WiFi,WheelchairAccessible	RestaurantsGoodForGroups
WheelchairAccessible,BikeParking,BusinessAcceptsCreditCards,free WiFi	RestaurantsGoodForGroups
BikeParking,casual,casual RestaurantsAttire,free WiFi,WheelchairAccessible	RestaurantsGoodForGroups
WheelchairAccessible,free WiFi	RestaurantsGoodForGroups
BikeParking,casual,free WiFi,RestaurantsTakeOut,WheelchairAccessible	RestaurantsGoodForGroups
BikeParking,casual RestaurantsAttire,free WiFi,RestaurantsTakeOut,WheelchairAccessible	RestaurantsGoodForGroups
BikeParking,free WiFi,RestaurantsTakeOut,WheelchairAccessible,BusinessAcceptsCreditCards	RestaurantsGoodForGroups

A natural next step might be to see which attributes are associated with families or in this case the attributes associated with 'RestaurantsGoodForGroups' and 'GoodForKids.'

The conclusions drawn from just looking at the attribute of groups are the same for families as well with the exception that top rated restaurants have outdoor seating whereas poorly rated restaurants aren't associated with them.

The lift among these items is approximately 1.01 implying independence among the antecedents and consequents.

For validation a very small subset was taken with the values computed manually compared to those of the algorithm

	BikeParking	BusinessAcceptsCreditCards	street
0	False	True	False
1	True	True	False
2	False	False	False
3	True	True	True
4	False	True	False
5	True	True	True
6	False	True	False
7	False	True	True
8	True	True	True
9	True	True	False

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(BikeParking)	(BusinessAcceptsCreditCards)	0.5	0.9	0.5	1.000000	1.111111	0.05	inf
1	(BusinessAcceptsCreditCards)	(BikeParking)	0.9	0.5	0.5	0.555556	1.111111	0.05	1.125

$$support(X) = \frac{\#\ of\ transactions\ with\ X}{\#\ of\ transactions\ in\ total}$$

$$row0 = \frac{5}{10} = 0.5 \qquad row1 = \frac{9}{10} = 0.9$$

The antecedent and consequent supports are calculated correctly given the formula and calculation.

$$support(X\ U\ Y) = \frac{\#\ of\ transactions\ with\ X\ and\ Y}{\#\ of\ transactions\ in\ total}$$

$$row0 = \frac{5}{10} = 0.5 \qquad row1 = \frac{5}{10} = 0.5$$

The supports for antecedent and consequent are calculated correctly as well.

$$conf(X \rightarrow Y) = \frac{support(X\ U\ Y)}{support(X)}$$

$$row0 = \frac{0.5}{0.5} = 1.0 \qquad row1 = \frac{0.5}{0.9} = 0.556$$

From this validation we can conclude that the algorithm from the library is implemented correctly.

K-means

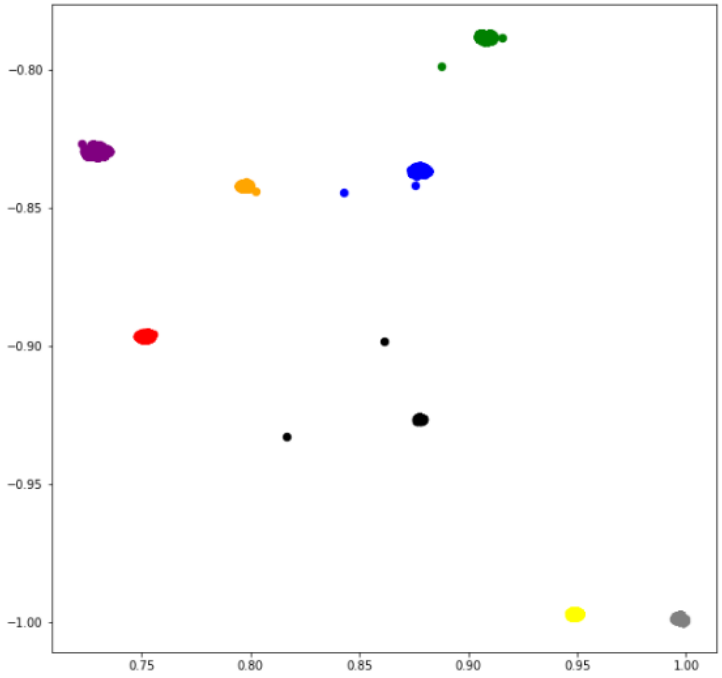
Another question we posed is do certain geographical locations have different distributions of their average ratings. To do this we got a table with longitude, latitude, and stars for each business.

SQL QUERY: SELECT longitude, latitude, stars FROM Businesses

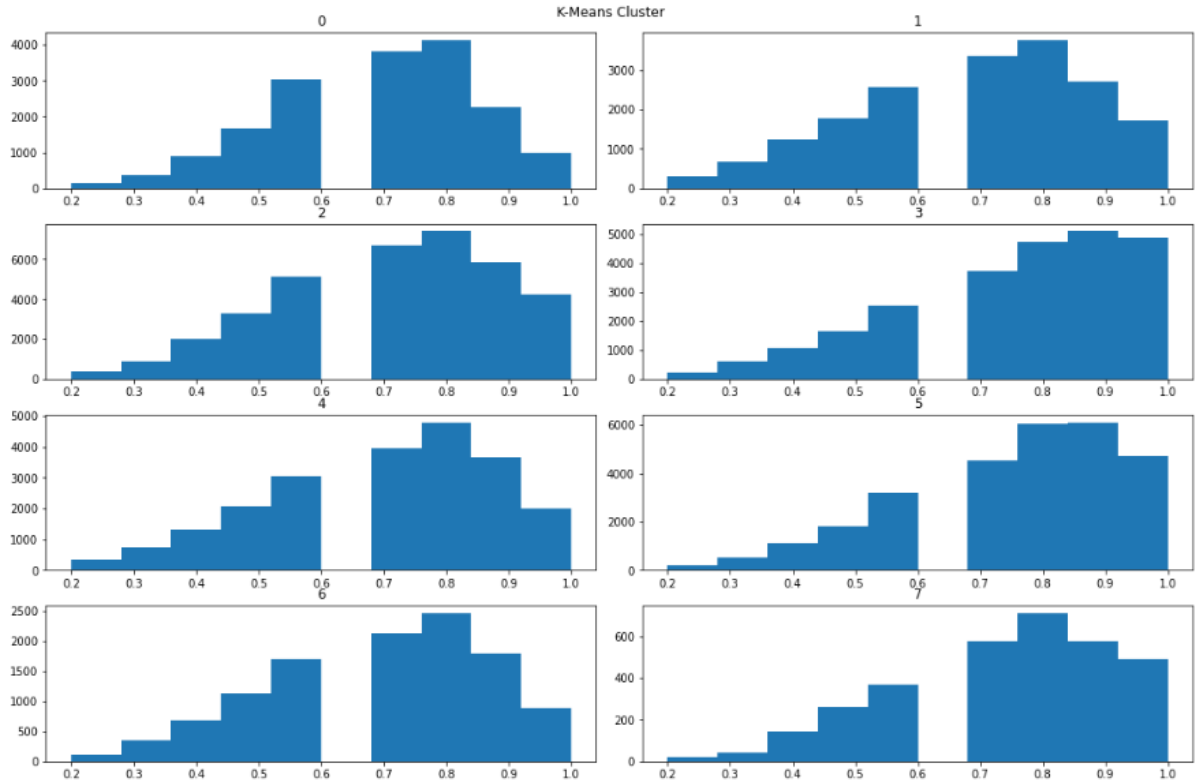
Each column was min-max normalized with the minimum value being added prior to reducing it to fit into the range of 0-1. The exact operations for processing the data are summarized below:

```
x = df_business[['latitude','longitude','stars']]
x.latitude = (x.latitude + x.latitude.min())
x.latitude = x.latitude / x.latitude.abs().max()
x.longitude = (x.longitude + x.longitude.min())
x.longitude = x.longitude / x.longitude.abs().max()
x.stars = x.stars / 5.0
x
```

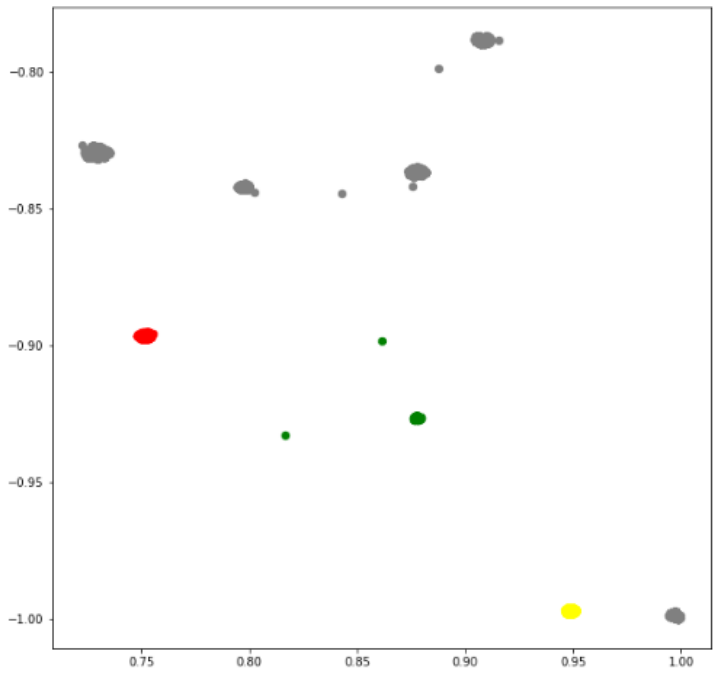
In this case we used k-means in which the number of clusters was empirically found to be 8. This was validated visually as shown below where each cluster represents its own separate color.



Originally all 3 factors were used in the k-means algorithm, but it was hard to visually interpret the 3-D clusters. Thus a 2-d approach was taken with clusters on longitude and latitude and then the distributions of each cluster was visually inspected.



The clusters 3,5 and 7 represent positively skewed statistics. Since we grouped by location, these clusters correspond to cities. They are represented as red, green, and yellow below.



The top 5 cities highlighted in clusters 3,5,7 from the graph represent the cities Austin, Portland, Vancouver, Boulder and Beaverton.

Further analysis should be done to compare why these cities have highly rated restaurants while others do not. Next steps may include understanding categories or attributes in these cities compared to others or augmenting socioeconomic factors to the dataset like salary to see if there are any other correlations. From prior experience these cities have major industries in them which could also mean people have money to spend therefore restaurants focus on customer experience.