# Yelp - Graph Based Analysis

Aryan Nath, Pankhi Mehta

May 2025

## 1 Research Focus

The main goal of our investigation of the Yelp dataset is to find patterns that show that the restaurant preferences of friends influence an individual's preferences.

The tools that we have used for our analysis are a recommendation system, plot-based analysis, link prediction algorithm, and geospatial analysis.

## 2 Database and Relationships

### 2.1 Database Filtering

Instead of reducing our dataset to just a city/state, in order to find region-agnostic patterns that reflect whether friends influence an individual's perspective, we decided to focus our filtering on retaining influential users and useful reviews.

We firstly removed users who had less than 5 fans, or less than a 100 review count, or less than 45 friends (based of the mean, median, and percentiles of theses parameters). We also removed people with fewer than 40 useful votes.

We removed businesses that have less than 44 reviews.
We then removed reviews if their writers or associated businesses were removed and if their useful rating is less than 8.

After this filtering, the entity count for each type came down to the following:

1. Users: 50241.
2. Businesses: 32115.
3. Reviews: 55519.

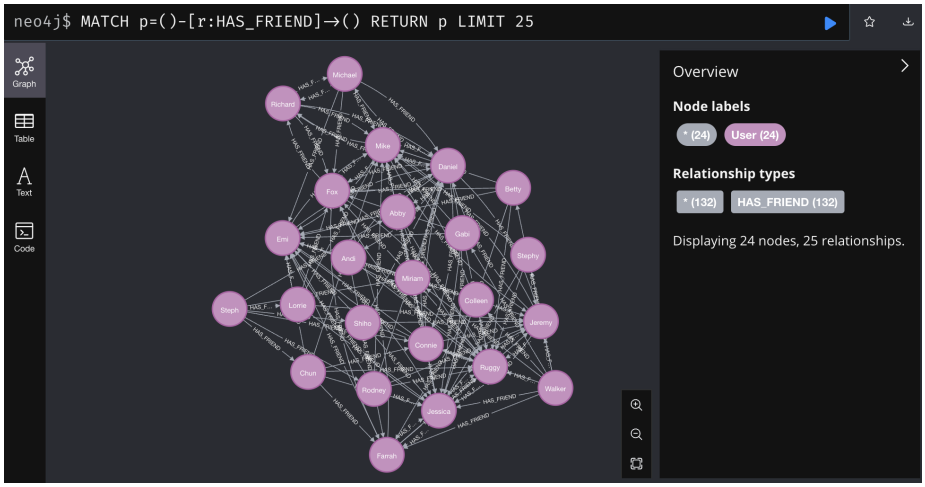
We created the following entities in our graph database:

1. User.
2. Business.
3. Category : represent the categories of businesses.
4. City : the city of a business.
5. State : the state of a business.
6. Review : the review written by a user.
7. Opinion : "Good", "Medium", "Bad" nodes used to discretize the star rating a review gives to a business.

## 2.2 Relationships

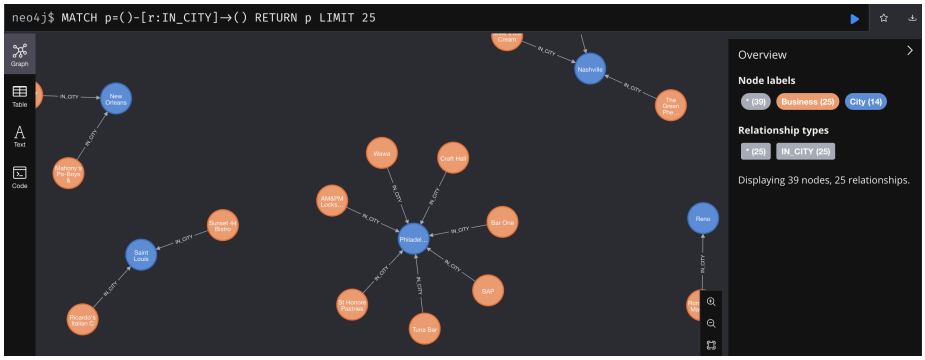
We made the following relationships in our graph database model:

1. HAS_FRIEND: User → User.

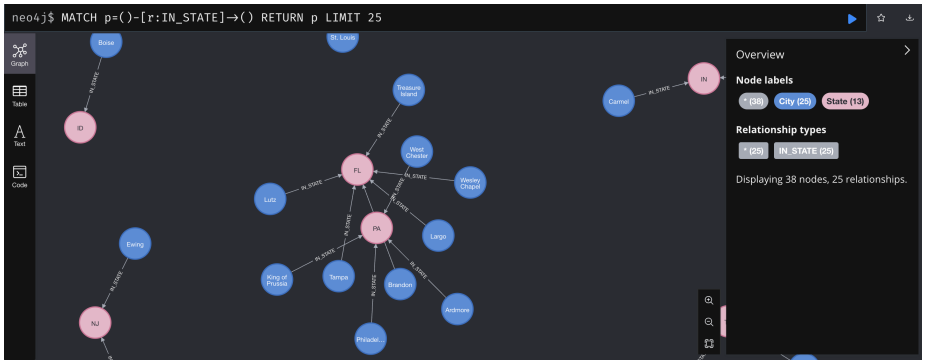



We reduced the number of friends for each user to 5 (the top 5 friends with the most fans) for our graph database so that creating this relationship would be feasible as many users had over a thousand friends.

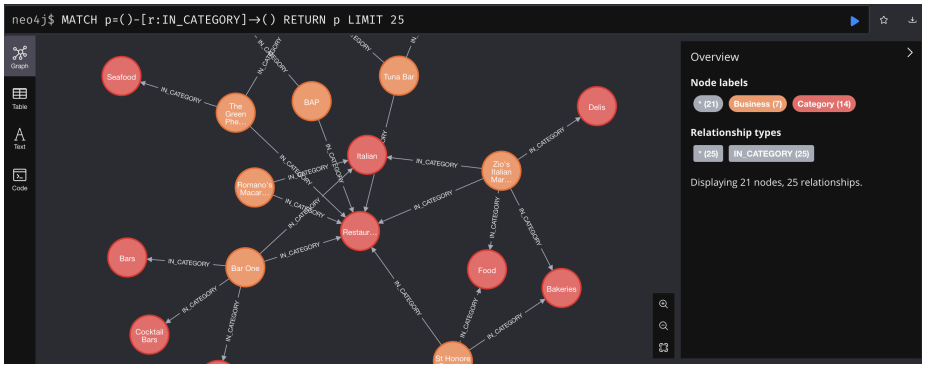
2. IN_CITY: Business → City.



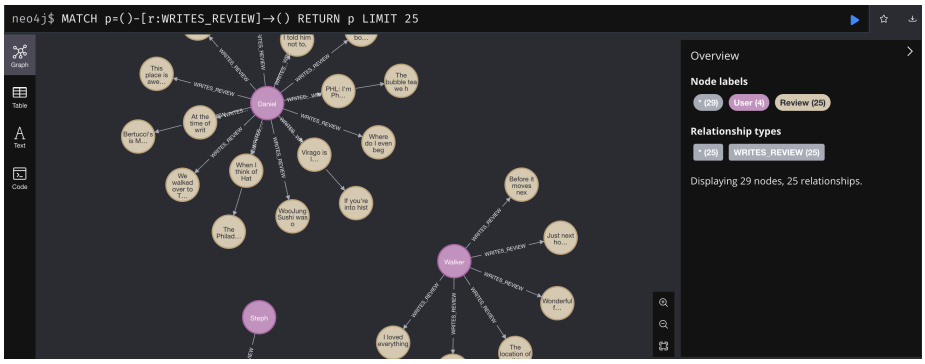

3. IN_STATE: Business → State.



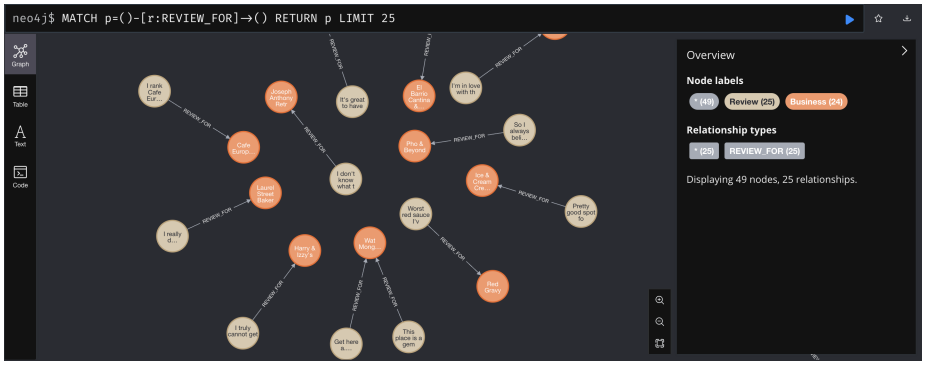

4. IN_CATEGORY: Business → Category.



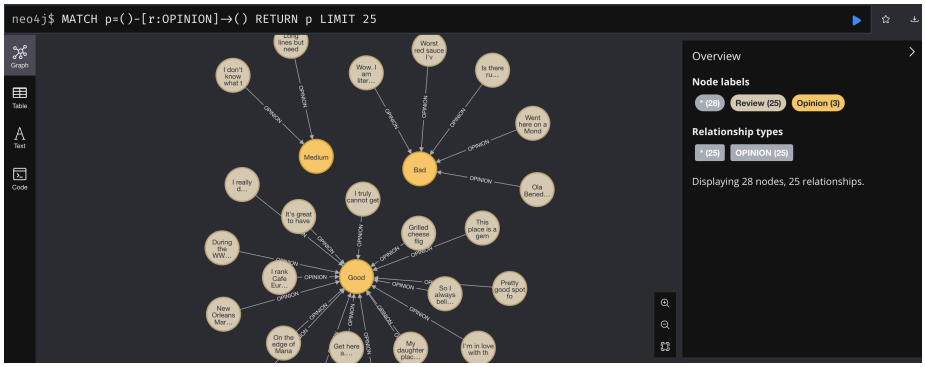
5. WRITES_REVIEW: User → Review.




6. REVIEW_FOR: Review → Business.




7. OPINION: Review → OPINION.

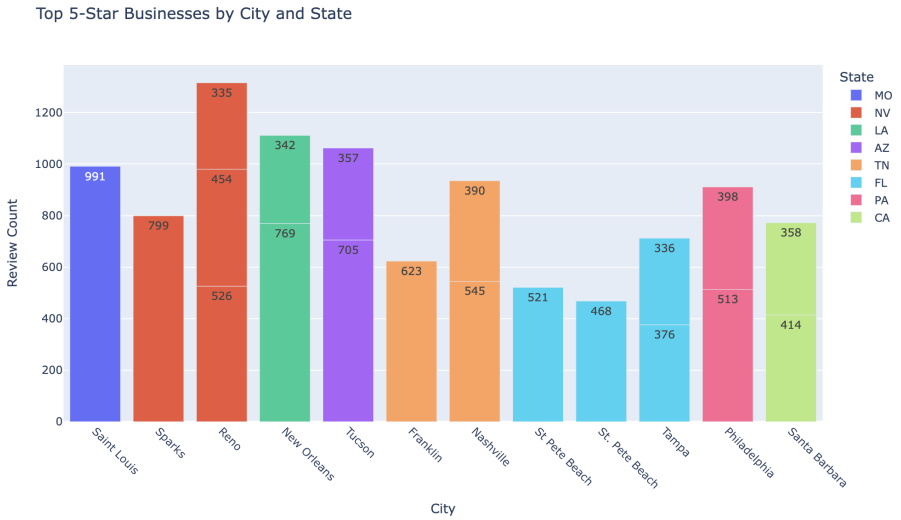


# 3 Database Exploration

In this section, we summarise some trends observed in our database using visualizations.

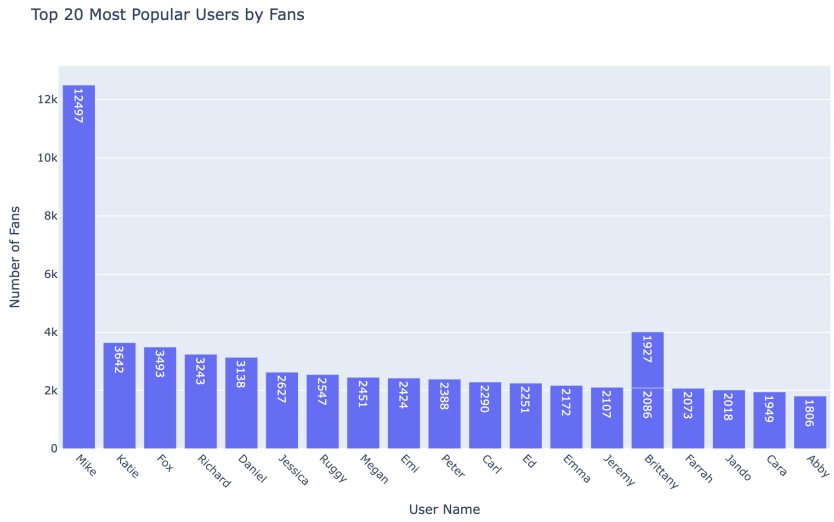
The following plot shows the top restaurants in the database, selected based on whether they have a 5 star rating and ordered by their review_count.


The city and state of these top restaurants are as follows:

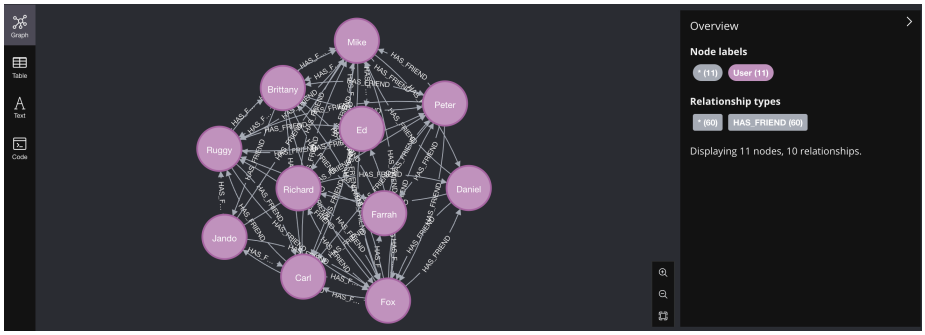


Here's a word cloud summarising the kind of reviews people have given the Blues City Deli restaurant:


The top users in the database, ordered by the size of their fanbase is as follows:



Here are the top 5 friends of based on the size of their fanbase:




We can see that Mike, Richard, Daniel, Ed, Carl, and Peter (Fox's friends) are also among the 20 most popular users in the Yelp database.

# 4 Recommmendation System: Is a user likely to eat the type of food their friends eat?

We use the restaurants visited by: 1) all friends of a user, 2) the top 5 most popular friends of a user, and compare the performance our recommendation system's recall for the business and categories of the food items that the user would like.

We have considered the user 'Fox' for our analysis. Firstly, we find all businesses that have been reviewed by Fox using:

```
MATCH (fox:User {name: "Fox"})-[:WRITES_REVIEW]->(review)-[:REVIEW_FOR]->(business)
MATCH (business)-[:IN_CATEGORY]->(category)
RETURN category.name as category,business.name as business,review.stars as review_stars
ORDER BY review.stars DESC
```

and store the set of all of these businesses and their categories in $fox\_businesses$ and $fox\_categories$.

The set of all visited businesses and their categories can be viewed here:
fox_businesses.txt
We then calculate the set of all businesses visited by Fox's friends in our graph database and their categories using the following code:

```
MATCH (fox:User {name: "Fox"})
MATCH (fox)-[:HAS_FRIEND]->(friend:User)
MATCH (friend)-[:WRITES_REVIEW]->(friendReview)-[:REVIEW_FOR]->(business:Business)
MATCH (friendReview)-[:OPINION]->(opinion)
WHERE opinion.name = "Good"
```

```
6
7  WITH fox, friend, business, COUNT(DISTINCT friendReview) AS positiveReviews
8  // WHERE NOT EXISTS {
9  //   MATCH (fox)-[:WRITES_REVIEW]->()-[:REVIEW_FOR]->(business)
10 // }
11
12 WITH business, COUNT(DISTINCT friend) AS friendCount, SUM(positiveReviews)
13 AS totalPositiveReviews
14 WITH business, friendCount, totalPositiveReviews,
15     (friendCount * 0.3) + (totalPositiveReviews * 0.7) AS recommendationScore
16 MATCH (business)-[:IN_CATEGORY]->(category)
17 WITH business, friendCount, totalPositiveReviews, recommendationScore,
18 COLLECT(category.name) AS categories
19 RETURN business.name AS businessName,
20     friendCount AS numberOfFriendsWhoLiked,
21     totalPositiveReviews AS totalPositiveReviews,
22     categories,
23     recommendationScore
24 ORDER BY recommendationScore DESC
```

We repeat the same with all friends of Fox in the original database, including the less popular ones:

```
1  import pandas as pd
2
3  df_friends = pd.read_csv('usersAllFriends.csv')
4
5  fox_friends = df_friends[df_friends['name'] == 'Fox']['friends'].values[0]
6  fox_friends_list = fox_friends.split(',')
7  print(f"Number of friends: {len(fox_friends_list)}")
8
9  def get_recommendations(tx, friend_ids):
10     query = """
11         UNWIND $friend_ids AS friend_id
12         MATCH (friend:User {user_id: friend_id})
13         MATCH (friend)-[:WRITES_REVIEW]->(friendReview)-[:REVIEW_FOR]->(business:Business)
14         MATCH (friendReview)-[:OPINION]->(opinion)
15         WHERE opinion.name = "Good"
16
17         WITH business, COUNT(DISTINCT friend) AS friendCount,
18         COUNT(DISTINCT friendReview) AS positiveReviews
19         WITH business, friendCount, positiveReviews,
20              (friendCount * 0.3) + (positiveReviews * 0.7) AS recommendationScore
21         MATCH (business)-[:IN_CATEGORY]->(category)
22         WITH business, friendCount, positiveReviews, recommendationScore,
23         COLLECT(category.name) AS categories
```

```
        RETURN business.name AS businessName,
               friendCount AS numberOfFriendsWhoLiked,
               positiveReviews AS totalPositiveReviews,
               categories,
               recommendationScore
        ORDER BY recommendationScore DESC
    """
    result = tx.run(query, friend_ids=friend_ids)
    return result.data()

with driver.session() as session:
    recommendations = session.execute_write(get_recommendations, fox_friends_list)

all_friends_categories = set()
all_friends_businesses = set()
for record in recommendations:
    all_friends_categories.update(record['categories'])
    all_friends_businesses.add(record['businessName'])
    print(record)
print("Categories:",all_friends_categories)
print("Businesses",all_friends_businesses)

driver.close()
```

We calculated the recommended business and their recommendation scores in the following steps:

1. First find all friends of Fox in the database using:

```
        MATCH (fox:User {name: "Fox"})
        MATCH (fox)-[:HAS_FRIEND]->(friend:User)
        MATCH (friend)-[:WRITES_REVIEW]->(friendReview)-[:REVIEW_FOR]->(business:Busi
```

2. Then we filtered the businesses to only the ones rated as "Good" by the friends and calculated the number of positive reviews per business and number of friends who liked each business using:

```
        MATCH (friendReview)-[:OPINION]->(opinion)
        WHERE opinion.name = "Good"
        WITH fox, friend, business, COUNT(DISTINCT friendReview) AS positiveReviews
        WITH business, COUNT(DISTINCT friend) AS friendCount, SUM(positiveReviews) AS
```

3. After this, we calculated the recommendation score using:

```
         WITH business, friendCount, totalPositiveReviews,
            (friendCount * 0.3) + (totalPositiveReviews * 0.7) AS recommendationScore
```

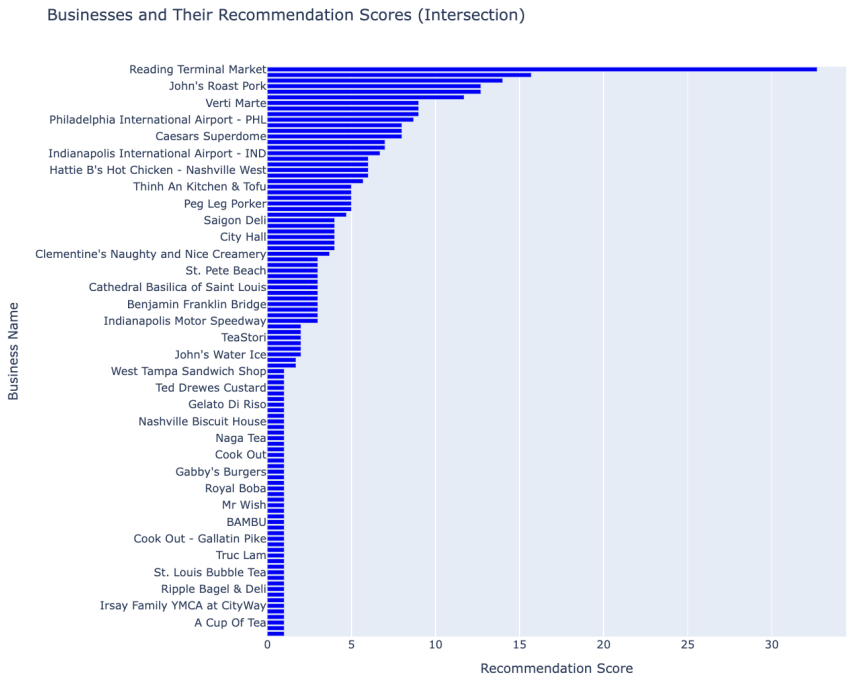
4. And retrieved the business categories using:

```
1           MATCH (business)-[:IN_CATEGORY]->(category)
2           WITH business, friendCount, totalPositiveReviews, recommendationScore, COLLEC
```

Finally, we computed the intersection of the restaurants/categories liked by Fox's friends and the restaurants that Fox has visited and computed the *recall* of our recommendation system by dividing the size of the intersection by the number of businesses visited/categories tried by Fox. The performance we achieved is as follows:

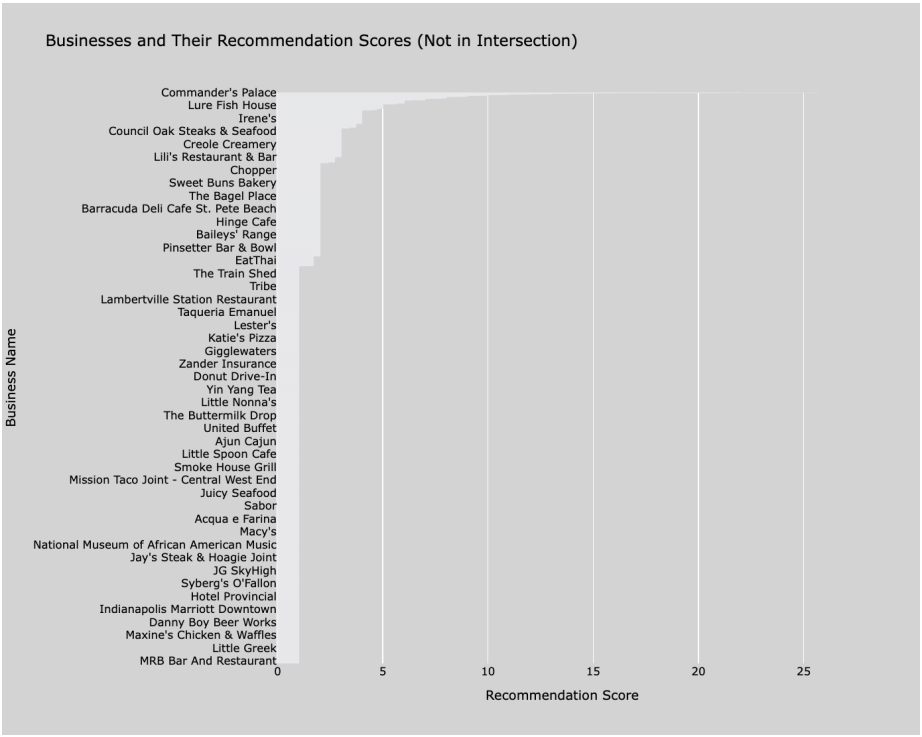
1. Recall for business recommendation using all friends: 0.75.
2. Recall for category recommendation using all friends: 1.0.
3. Recall for business recommendation using top 5 most influential friends: 0.176.
4. Recall for category recommendation using top 5 most influential friends: 0.935064935064935.

The recommendation scores of the businesses in the intersection are as follows:




It can be observed that a majority of the businesses in the intersection have recommendation scores $> 1$.

The recommendation scores for businesses not in the intersection are as follows:




(the bar plot color could not be changed for a large number of y-axis labels) Hence, even though there are some businesses not in the intersection of businesses that Fox has visited and the ones that have been rated as good by Fox's friends with non negligible recommendation score, most of these businesses have a recommendation score of 1 (lowest).

Therefore, there is a clear pattern that the businesses that have been rated as good by Fox's friends are more likely to be visited by Fox and that a friend having a popular fanbase (top 5 most popular friends) do not influence Fox's preferences as much as all of her friends collectively do.

## Use cases of the recommendation system

Based on what we have stated in our above analysis, this recommendation system can be used by users to filter down on the businesses to the ones they will like with a higher probability.

# 5 Link prediction: Do people visiting the same places tend to be friends?

To elaborate on our hypothesis that friends might have similar preferences, we implemented a link prediction model to predict the [:HAS_FRIEND] relation using the Adamic adar score based on common businesses. We also took into account preferences such that the review ratings hat to be within $\pm 1$ stars of each other, so that we ensured that common businesses were only taken if both people shared a similar opinion on them. The accuracy of this link predicting model on finding yelp friends may inform us on how much visiting/reviewing the same places plays a part in yelp friendships.

To make calculating this easier, we first created a copy of the database and converted review nodes into relations between Users and Businesses instead, using the following cypher command:

```
MATCH (a:User) -[r1:WRITES_REVIEW]-> (b:REVIEW) -[r2:REVIEW_OF]-> (c:Business)
WITH a,b,c
CREATE (a)-[r:REVIEWS]->(c)
SET r = b
//we set r to have the same properties as b
WITH b
Detach delete b
```

To run the algorithm, we used the neo4j python API. We first selected random Users who have reviewed any business:

```
MATCH (a)-[:REVIEWS]->()
RETURN a.user_id, rand() as r
ORDER BY r Limit $n
```

And defined 10 predicted friends for each, based on users which had the highest AA index with respect to the node in question.

```
 MATCH (n1:User {user_id:$user})-[r1:REVIEWS]->(b:Business)<-[r2:REVIEWS]-(n2:User)
WHERE -2 < (r1.stars - r2.stars) < 2 AND NOT (n1)-[:HAS_FRIEND]->(n2)

//Here we have ensured that n1 and n2 are not already friends in our filtered graph,
//and that they have similar opinions on b

WITH n1, n2, b
match  (b)--()
with n1,n2, count(*) AS bDegree
WITH n1, n2, SUM(1.0 / log(bDegree)) AS adamicScore

MATCH (n1)-[:REVIEWS]->(:Business)-[:IN_CATEGORY]->(c:Category)
<-[:IN_CATEGORY]-(:Business)<-[:REVIEWS]-(n2)
```

```
    WITH n1, n2, c, adamicScore, count(*) as sharedReviewCount
    match (c)--()
    WITH n1, n2, c, adamicScore, sharedReviewCount,count(*) AS cDegree

    WITH n1, n2, adamicScore, SUM(1.0*sharedReviewCount / cDegree) AS categoryBonus

//To further account for posssible similarity in preferences between
//friends, we added a "bonus" based on the AA score according to the
//categories of business that n1 and n2 review. Here sharedReviewCount
//is incroperted such that each time user reveiws a buisiness in c, it
//is counted as its own seperate edge berween user and c

    RETURN adamicScore + categoryBonus AS totalScore,
    n2.user_id order by totalScore desc limit 10
```

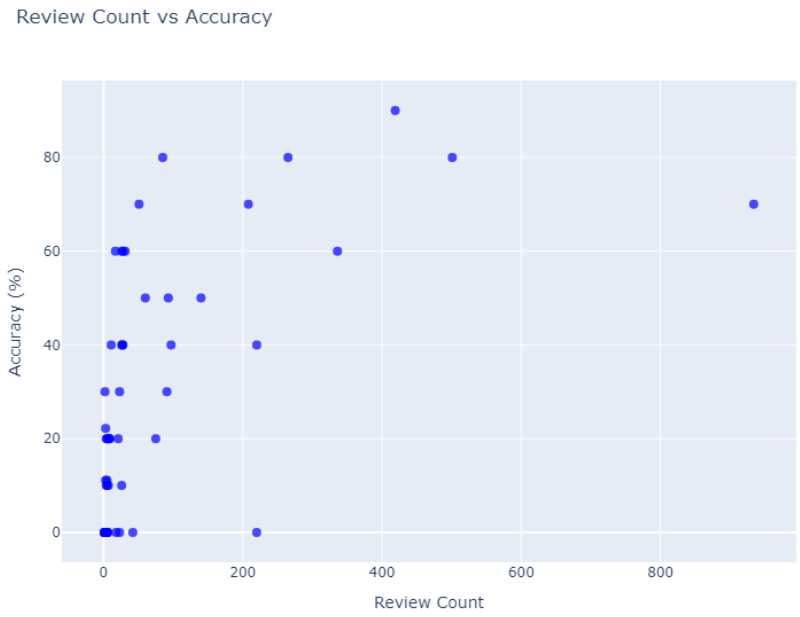
With the list of predicted friends, we checked the accuracy of the predictions by seeing what proportion of these appeared in the original friend list of the user in question, that is the friend list prior to our restricting [:HAS FRIEND] to the top 5 most popular friends.
The result of total correct predictions when run on 50 random users was 50.56%.

This score is not ideal, indicating that there may be more to yelp friends than similar business preferences. People may be friends with those they know in real life, including those who live far away and thus cannot review the same businesses and so on.
It may also indicate a pitfall of our neo4j graph filtering: in filtering out business nodes, businesses that were removed during filtering may have been businesses that are common among certain friends, thus clouding the algorithms performance.

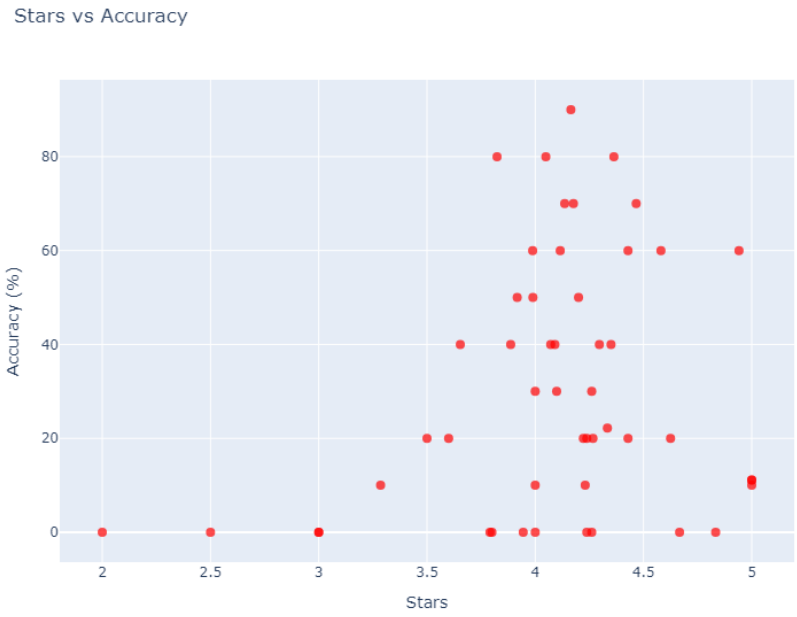
That being said, an accuracy of 45.5% is not inconsequential:
when looking at the distribution of number of friends (prior to filtering to 5 most popular) we see that the mean and median number of friends per user are 78 and 29 respectively. Out of the roughly 50,000 total users, that means that if one were to predict friends randomly, the probability of a correct prediction would be around 0.16% going by the mean or 0.06% going by the median number of friends per User. Even for the user with the most number of friends (5670 friends), a random prediction would only be right about 11% of the time, which is 4 times less accurate than the link predictor model.
Thus, this prediction method is somewhat useful for Yelp users and Yelp itself, since it can decently predict who might be friends or at the very least suggest users to accounts which they share opinions, so that they can look at the suggested accounts' reviews when looking for potential establishments to visit.

Once again running the prediction on 50 random nodes and plotting both average stars given by user, and number of reviews given by user against accuracy,

we got the following graphs:




This shows that the accuracy of this link prediction is proportional to the number of reviews a user has given. This makes sense, since the more reviews that a user has given, the more potential common neighbors they can have with other users. This would mean that the prediction algorithm has more information to go off of, intern leading to more accuracy.
Additionally, this trend might also suggest that users who give many reviews may tend to have more friends.
To test this, we calculated the correlation of yelp friends and the review count number using the csv data obtained prior to filtering friends lists. The resultant Pearson positive correlation of 0.41, which is statistically significant with a significance of $\alpha = 0.01$. Though the direction of causality for this correlation is unclear, logic suggests that writing more reviews may lead to you gaining more yelp friends.



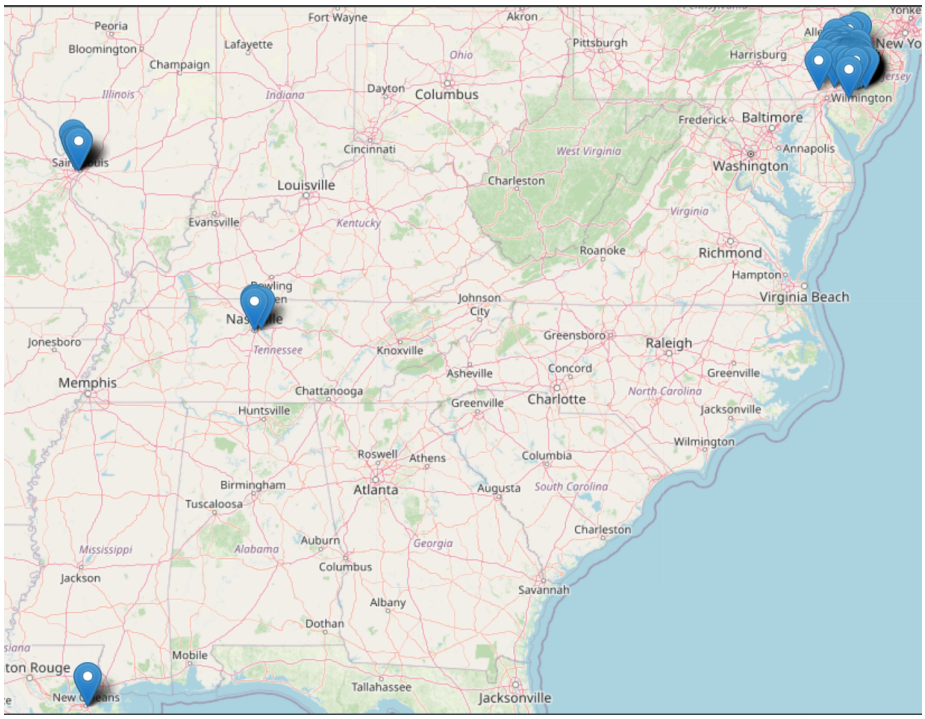

This figure seems to show that the average star rating given out by a user has little to do with the accuracy of the link predictor, as long as the star rating is above 3. When looking at the data, we see that majority of the ratings are good (above 3), and given that the link predictor requires users to give similar ratings to the same business in order count it as a common neighbor, it makes sense that those who on average rate 3 or below would not have an accurate prediction, since they may be less likely to find similarly rated reviews on the same businesses.

| Opinion | number_of_reviews |
|---|---|
| "Bad" | 6814 |
| "Medium" | 6537 |
| "Good" | 42168 |

# 6 Location and time analysis using clustering

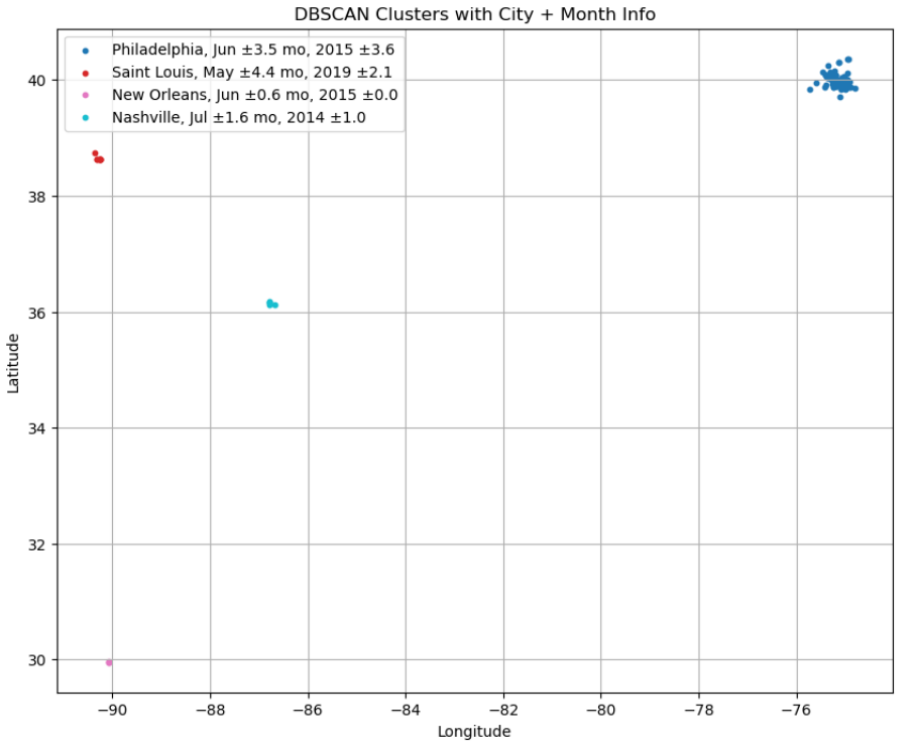
Given the locations of the businesses a person has reviewed and when they had reviewed them, we wanted to see what can be deduced about where that person lives and travels, and when.
For this, we only looked into the Yelper with the most reviews in our database, a person named Michelle.
First we obtained the coordinates of all the businesses she reviews and when, using the query bellow, and plotted the longitudes and latitudes onto a map using the folium python library.

```
Match (n:User {user_id:$user})-[r:REVIEWS]->(b:Business)
with [b.longitude,b.latitude] as coordinates, b,r
return coordinates, r.date, b.city
```




We can see that these form clear clusters in certain areas, indicating places Michelle has lived or visited.
To analyses this further, we used DBSCAN to obtain the clusters, since it does not require any input on number of clusters nor does it require clusters be of any uniform density or shape, so it can be generalised to be used on any User.
We used the city names of the businesses and date information from the corresponding reviews to find out generally where and when Michelle was in these areas. To do this, we took the mode city for each cluster, and the average month

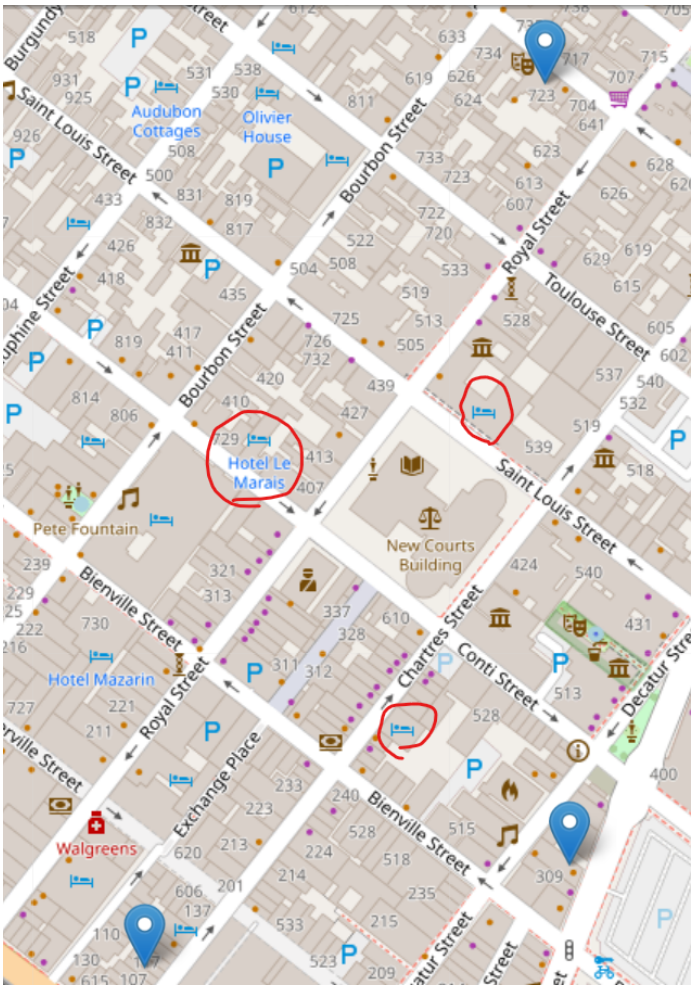
and years for the reviews, along with the standard deviations of the same. This was the result:




Additionally, analysing clusters shows us that Michelle had:
918 reviews from the Philadelphia area, from 2008 to 2021
8 reviews from the Saint Louis area, from 2014 to 2020
3 reviews from the New Orleans area, from 2015 to 2015
5 reviews from the Nashville area, from 2013 to 2015
Reviews from New Orleans are between May to Sep of 2015

From the cluster data of year and month, and cluster density shown on graph it can be deduced that Michelle most likely lives in Philadelphia, since majority of her reviews are in that city, over many years, and year round.
The data also suggests that this person visits the Saint Louis area frequently, since they have written reviews in that area over multiple years in that area, at different times of year.
Similarly, due to the standard deviation in years and months of reviews for New Orleans, we can deduce that Michelle likely only visited New Orleans once around June of 2015 for a some duration of time, at the max 5 months, but likely less than that considering she may have submitted reviews after her visit. In fact, if you were to zoom in on the map of the area of these New Orleans

reviews, you see that the following hotels are close to all 3 of these business locations: *Hotel Le Maraid, Omni Royal Orleans and Hotel de la Poste - French Quarter* are all situated roughly in between the 3 locations, suggesting these are possibly where Michelle was staying at this time under t4he assumption that she would've went to businesses generally close to where she was staying.




Michelle likely visited Nashvil multiple times around 2013-15. It is also possible that they lived in Nashvil for a continuous stretch sometime from 2013-2015, but the low variance in the months of reviews and the fact that she was still writing reviews in Philidelphia in those years suggest otherwise. By virtue of the review she had written for the business, we also know she stayed at the hotel *Hyatt Place Nashville Downtown* in 2015. Note how this hotel is located generally in between all the other locations she had reviewed in Nashville.

Next, we tried to find what the most common category of business Michelle

was reviewing for each cluster (by running adding collect(category.name) to our analysis), which gave us:

Philadelphia area - Most common categories:
    Restaurants: 731
    Food: 320
    Nightlife: 308
    Bars: 291
    American (New): 212
Saint Louis area - Most common categories:
    Restaurants: 7
    Bars: 3
    Nightlife: 3
    Food: 2
    Breweries: 2
New Orleans area - Most common categories:
    Restaurants: 3
    Cocktail Bars: 2
    Bars: 2
    Nightlife: 2
    American (Traditional): 2
Nashville area - Most common categories:
    Bars: 3
    Nightlife: 3
    Restaurants: 2
    Lounges: 1
    Local Flavor: 1

This tells us that Michelle tends to review food and drink related businesses, and probably likes "to go out on the town", as the saying goes, considering the high occurrences of the Bars and Nightlife categories.

This example shows how the review data for a user may be used to estimate their travel, when they tend to be in a location or when they tend to visit businesses. All of this information can be useful to the restaurants/businesses, since accumulating this data over many users may allow them to predict when and where people will be visiting establishments.
They can do this by creating similar clusters as described here clusters, but across multiple users, along the temporal axis as well as spatial coordinate, and only on the categories of their interest.

This sort of information, when collected on a specific person such as Michelle can also be used for investigative purposes.