CS 838 (Spring 2017) - Data Science

Project Stage - 5 Report

Group : 17

Gautam Singh

Harsh Singhal

Naman Agrawal

Objective : Data Analysis

The objective of this stage was data analysis on the integrated and cleaned table obtained from previous stage.

1. Statistics of Table E:

Schema of Table E -

< ID, Name, Phone, Zipcode, State, City, Address, Delivery, Takeout, Outdoor_seating, pricy, rating >

PS: We have regenerated the merged table and added two more fields for analysis. Code for merging is given below along with code for data analysis.

Some sample tuples from table E are as follows:

id	1	2	3	4	5
restaurant_name	McSorleys Old Ale House	El Techo de Lolinda	Phat Philly Cheesesteaks	Samovar Tea Lounge	Bean Bag Cafe
phone	(212) 473-9148	(415) 550-6970	(415) 550-7428	(415) 227-9400	(415) 563-3634
zipcode	10003	94110	94110	94103	94117
state	NY	CA	CA	CA	CA
city	New York	San Francisco	San Francisco	San Francisco	San Francisco
address	15 E 7th Street	2518 Mission District Street	3388 24th Street	730 Howard Street	601 Divisadero Street
delivery	0	0	0	0	0
takeout	0	0	1	1	0
outdoor_seating	0	1	1	1	1
pricy	1	2	1	2	1
rating	4.4	3.4	3.5	3.7	3.6

Price denotes how pricey the restaurant is.

Rating is from 1 to 5.

The meaning of the other attributes is self-explanatory.

Number of Tuples in E - 275

Link to this processed table – Table E

2. Data Analysis Task:

We have used OLAP-based SQL queries for analysis.

SQlite was used to write SQL queries for analyzing the db. We have used some interesting queries as below for drawing analysis:

a. impact of delivery facility on pricing and rating

b. impact of takeout facility on pricing and rating

c. impact of outdoor seating facility on pricing and rating
d. impact of city on pricing and rating
e. impact of location on pricing and rating
f. number of pricy restaurants in each city
g. drill down on chicago's expensive restaurants.

CODE

a. Code to merge two new columns into the merge table

```
# coding: utf-8

# In[34]:

import py_entitymatching as em #Import megallan entity matching
library
import math
import warnings
warnings.filterwarnings('ignore')


# In[35]:

def phone_match(str1,str2):
    if type(str1) is float and type(str2) is float:
        if math.isnan(str1) and math.isnan(str2):
            return True
    elif type(str1) is float:
        if math.isnan(str1):
            return False
    elif type(str2) is float:
        if math.isnan(str2):
            return False
    else:
        stra = ""
        strb = ""
        for ch in str1:
            if ch.isdigit():
                stra += ch
        for ch in str2:
            if ch.isdigit():
                strb += ch
        if stra == strb:
            return True
        else:
            return False


# In[36]:

matching_records = em.read_csv_metadata("before_merging.csv",key="id")
yelp_original = em.read_csv_metadata("yelp_original.csv",key="id")
zomato_original = em.read_csv_metadata("zomato_original.csv",key="id")
```

```
# In[37]:

matching_records.columns


# In[38]:

yelp_original.columns # Original CSV file with added Pricyness column


# In[39]:

zomato_original.columns # Original CSV with added Rating column


# ## Schema Merging

# In[41]:

indexes_to_keep = set()
index = 0

for index in range(matching_records.shape[0]):
    tuple = matching_records.iloc[index]
    if tuple['predicted'] == 1:

        # Merging the Names -
        # Picking the one that has more length
        if len(tuple['ltable_Name']) > len(tuple['rtable_Name']):
            tuple['restaurant_name'] = tuple['ltable_Name']
        else:
            tuple['restaurant_name'] = tuple['rtable_Name']

        # Merging the Phone no -
        phone1 = tuple['ltable_Phone']
        phone2 = tuple['rtable_Phone']
        if phone_match(phone1, phone2) is True: # When phone numbers
are same
            tuple['phone'] = phone1
        else: # Case when phone nos are different. We keep both
separated by comma.
            tuple['phone'] = phone1+ "," + phone2

        # Merging the Zipcode -
        # Since blocking was done based on exact match for ZipCode,
picking the left table attribute value
        tuple['zipcode'] = tuple['ltable_Zipcode']
```

```
        # Merging the State -
        # Picking the left table attribute
        tuple['state'] = tuple['ltable_State']

        # Merging the City -
        # Picking the left table attribute
        tuple['city'] = tuple['ltable_City']

        # Merging the Address
        # Picking the one that has more length
        if len(tuple['ltable_Address']) >
len(tuple['rtable_Address']):
            tuple['address'] = tuple['ltable_Address']
        else:
            tuple['address'] = tuple['rtable_Address']


        tuple['delivery'] = tuple['rtable_Delivery']
        tuple['takeout'] = tuple['rtable_Takeout']
        tuple['outdoor_seating'] = tuple['ltable_Outdoor_seating']

        # For rating
        for ind in range(zomato_original.shape[0]):
            entry = zomato_original.iloc[ind]
            if entry['Name'] == tuple['rtable_Name']:
                tuple['rating'] = entry['Rating']
                break

        # For priciness
        for ind in range(yelp_original.shape[0]):
            entry = yelp_original.iloc[ind]
            if entry['Name'] == tuple['ltable_Name']:
                tuple['pricy'] = entry['Price']
                break

        # Updating the tuple in predications table
        matching_records.iloc[index] = tuple
        indexes_to_keep.add(index)

    index += 1


# In[42]:

# Print the schema
matching_records.head(1)


# In[44]:
```

```
# Fetch only those rows where predicted = "1" => get correctly matched
tuples
sliced = matching_records.take(list(indexes_to_keep))

# Drop columns before merging.
# Dropping old attributes

del sliced['ltable_Name']
del sliced['rtable_Name']
del sliced['ltable_Phone']
del sliced['rtable_Phone']
del sliced['ltable_Zipcode']
del sliced['rtable_Zipcode']
del sliced['ltable_State']
del sliced['rtable_State']
del sliced['ltable_City']
del sliced['rtable_City']
del sliced['ltable_Address']
del sliced['rtable_Address']
del sliced['ltable_Delivery']
del sliced['rtable_Delivery']
del sliced['ltable_Takeout']
del sliced['rtable_Takeout']
del sliced['ltable_Outdoor_seating']
del sliced['rtable_Outdoor_seating']
del sliced['Label'] # Dropping the column'Label'
del sliced['predicted'] # Dropping the column 'predicted

sliced.to_csv("filtered_predictions.csv") # Writing the resultant
table to a CSV file.s
```

b. Code for Data Analysis

```
# In[1]:

import py_entitymatching as em #Import megallan entity matching
library
import math
import warnings
warnings.filterwarnings('ignore')

# Pandas has useful data-structure and analysis tool
import pandas as pd
# numpy has a lots of useful math related modules
import numpy as np
# to check execution time
import time
# Helpful function to display intermittent result
from IPython.display import display


# In[39]:

matching_records =
em.read_csv_metadata("filtered_predictions.csv",key="id")
yelp_original = em.read_csv_metadata("yelp_original.csv",key="id")
zomato_original = em.read_csv_metadata("zomato_original.csv",key="id")
matching_records.head(1)


# ### Load the dataset

# In[3]:

#sqlalchemy for sqlite3 use
from sqlalchemy import create_engine
import sqlite3
#create a database where we'll load the dataset from the csv file
engine = create_engine('sqlite:///yelpzomato.db')
connection = engine.connect()


# In[4]:

import string

for data in pd.read_csv("filtered_predictions.csv",
 iterator=True, encoding='utf-8'):
```

```
    data.to_sql('data', engine, if_exists='append')


# ### impact of delivery facility on pricing and rating

# In[5]:

a = connection.execute("select AVG(rating) from data where delivery =
0")
b = connection.execute("select AVG(rating) from data where delivery =
1")
for data in zip(a,b):
    print data


# #### Insight -
# Takeout facility hasn't affected the rating significantly.

# In[6]:

a = connection.execute("select AVG(pricy) from data where delivery =
0")
b = connection.execute("select AVG(pricy) from data where delivery =
1")
for data in zip(a,b):
    print data


# #### Insight -
# Restaurants that do not offer delivery facility on average are more
pricy.

# ### impact of takeout facility on pricing and rating

# In[7]:

a = connection.execute("select AVG(rating) from data where takeout =
0")
b = connection.execute("select AVG(rating) from data where takeout =
1")
for data in zip(a,b):
    print data


# #### Insight -
# Takeout facility hasn't affected the rating significantly, which
means that perhaps people do not care much about outdoor seating

# In[8]:
```

```
a = connection.execute("select AVG(pricy) from data where takeout =
0")
b = connection.execute("select AVG(pricy) from data where takeout =
1")
for data in zip(a,b):
    print data


# #### Insight -
# Restaurants that do not offer takeout facility on average are more
pricy.

# ### impact of outdoor seating on pricing and rating

# In[9]:

a = connection.execute("select AVG(rating) from data where
outdoor_seating = 0")
b = connection.execute("select AVG(rating) from data where
outdoor_seating = 1")
for data in zip(a,b):
    print data


# #### Insight -
# Outdoor seating hasn't affected the rating significantly, which
means that perhaps people do not care much about
# outdoor seating.

# In[30]:

a = connection.execute("select AVG(pricy) from data where takeout =
0")
b = connection.execute("select AVG(pricy) from data where takeout =
1")
for data in zip(a,b):
    print data


# #### Insight -
# Restaurants that do not offer outdoor seating on average are more
pricy.

# ### impact of delivery and takeout on rating and pricing

# In[11]:

a = connection.execute("select AVG(rating) from data where delivery =
0 and takeout = 0")
b = connection.execute("select AVG(rating) from data where delivery =
```

```
0 and takeout = 1")
c = connection.execute("select AVG(rating) from data where delivery =
1 and takeout = 0")
d = connection.execute("select AVG(rating) from data where delivery =
1 and takeout = 1")
for data in zip(a,b,c,d):
    print data


# In[12]:

a = connection.execute("select AVG(pricy) from data where delivery = 0
and takeout = 0")
b = connection.execute("select AVG(pricy) from data where delivery = 0
and takeout = 1")
c = connection.execute("select AVG(pricy) from data where delivery = 1
and takeout = 0")
d = connection.execute("select AVG(pricy) from data where delivery = 1
and takeout = 1")
for data in zip(a,b,c,d):
    print data


# ### impact of delivery and outdoor seating on rating and pricing

# In[13]:

a = connection.execute("select AVG(rating) from data where delivery =
0 and outdoor_seating = 0")
b = connection.execute("select AVG(rating) from data where delivery =
0 and outdoor_seating = 1")
c = connection.execute("select AVG(rating) from data where delivery =
1 and outdoor_seating = 0")
d = connection.execute("select AVG(rating) from data where delivery =
1 and outdoor_seating = 1")
for data in zip(a,b,c,d):
    print data


# In[14]:

a = connection.execute("select AVG(pricy) from data where delivery = 0
and takeout = 0")
b = connection.execute("select AVG(pricy) from data where delivery = 0
and takeout = 1")
c = connection.execute("select AVG(pricy) from data where delivery = 1
and takeout = 0")
d = connection.execute("select AVG(pricy) from data where delivery = 1
and takeout = 1")
for data in zip(a,b,c,d):
```

```
    print data


# ### impact of takeout and outdoor seating on rating and pricing

# In[15]:

a = connection.execute("select AVG(rating) from data where takeout = 0
and outdoor_seating = 0")
b = connection.execute("select AVG(rating) from data where takeout = 0
and outdoor_seating = 1")
c = connection.execute("select AVG(rating) from data where takeout = 1
and outdoor_seating = 0")
d = connection.execute("select AVG(rating) from data where takeout = 1
and outdoor_seating = 1")
for data in zip(a,b,c,d):
    print data


# In[16]:

a = connection.execute("select AVG(pricy) from data where takeout = 0
and outdoor_seating = 0")
b = connection.execute("select AVG(pricy) from data where takeout = 0
and outdoor_seating = 1")
c = connection.execute("select AVG(pricy) from data where takeout = 1
and outdoor_seating = 0")
d = connection.execute("select AVG(pricy) from data where takeout = 1
and outdoor_seating = 1")
for data in zip(a,b,c,d):
    print data


# ### impact of delivery, takout and outdoor seating on rating and
pricing

# In[17]:

a = connection.execute("select AVG(rating) from data where delivery =
0 and takeout = 0 and outdoor_seating = 0")
b = connection.execute("select AVG(rating) from data where delivery =
0 and takeout = 0 and outdoor_seating = 1")
c = connection.execute("select AVG(rating) from data where delivery =
0 and takeout = 1 and outdoor_seating = 0")
d = connection.execute("select AVG(rating) from data where delivery =
0 and takeout = 1 and outdoor_seating = 1")
e = connection.execute("select AVG(rating) from data where delivery =
1 and takeout = 0 and outdoor_seating = 0")
f = connection.execute("select AVG(rating) from data where delivery =
1 and takeout = 0 and outdoor_seating = 1")
```

```
g = connection.execute("select AVG(rating) from data where delivery =
1 and takeout = 1 and outdoor_seating = 0")
h = connection.execute("select AVG(rating) from data where delivery =
1 and takeout = 1 and outdoor_seating = 1")
for data in zip(a,b,c,d,e,f,g,h):
    print data


# #### Interesting Insight -
#
# + Customers prefer restaruants that have delivery and takeout
facility and rate them the highest.
# + Restaurants that offer all three services overall have received
lowest average rating, perhaps since they are cheapest,
# they do not live upto the standard of the costlier peers in the
game.
# + Other categories lie in the middle.

# In[18]:

a = connection.execute("select AVG(pricy) from data where delivery = 0
and takeout = 0 and outdoor_seating = 0")
b = connection.execute("select AVG(pricy) from data where delivery = 0
and takeout = 0 and outdoor_seating = 1")
c = connection.execute("select AVG(pricy) from data where delivery = 0
and takeout = 1 and outdoor_seating = 0")
d = connection.execute("select AVG(pricy) from data where delivery = 0
and takeout = 1 and outdoor_seating = 1")
e = connection.execute("select AVG(pricy) from data where delivery = 1
and takeout = 0 and outdoor_seating = 0")
f = connection.execute("select AVG(pricy) from data where delivery = 1
and takeout = 0 and outdoor_seating = 1")
g = connection.execute("select AVG(pricy) from data where delivery = 1
and takeout = 1 and outdoor_seating = 0")
h = connection.execute("select AVG(pricy) from data where delivery = 1
and takeout = 1 and outdoor_seating = 1")
for data in zip(a,b,c,d,e,f,g,h):
    print data


# #### Interesting Insight -
#
# + The priciest restaurants do not offer delivery, takeout and
outdoor seating facility. They belong to luxurious class.
# + The next category is has the restaurants which are slightly less
pricy but just offer outdoor seating facility.
# + The trend goes on like this and the restaurants which offer all
three facilites are the cheapest.

# ### Data Distribution based on city
```

```
# In[19]:

a = connection.execute("select city, count(city) from data group by
city")
for data in a:
    print data


# In[ ]:

Brooklyn has very few instances, hence any inference drawn might be
inaccurate.


# ### impact of city on rating and pricing

# In[20]:

a = connection.execute("select city, AVG(pricy), AVG(rating) from data
group by city")
for data in a:
    print data


# + Shows that Seattle is least pricy but has overall average user
rating.
# + Restaurants in San Francisco have overall worst rating.
# + New York has pricest restaurants with highest user rating.
# + Brooklyn - no inference due to lack to adequate number of
instances.

# ### Restaurants in Chicago which are most expensive

# In[27]:

a = connection.execute("select restaurant_name,pricy from data where
city = 'Chicago' and pricy = 4")
for data in a:
    print data


# ### Number of most pricy(4) restaurants in each city

# In[29]:

a = connection.execute("select count(restaurant_name),city from data
where pricy = 4 group by city")
for data in a:
    print data
```

```
# + The above result suggests that 'New York' has over all more number
of pricy restaurants.
# + Based the above result and also the data distribution city wise,
we observe that, Seattle is the cheapest of all followed by San
Francisco, then Chicago and New York being the most expensive one.
# + Brooklyn has not been considered since there are very few
instances of Brooklyn.

# ### Impact of Location on Price and Rating

# In[34]:

a = connection.execute("select min(zipcode), max(zipcode), city from
data group by city")
for data in a:
    print data


# + The above query gives the range of zipcode for restaurant in
records for each city.
# + This data can be used for location based analysis.

# ### Location based analysis on New York

# In[38]:

a = connection.execute("select
avg(pricy),avg(rating),count(restaurant_name) from data where zipcode
between 10001 and 10005")
b = connection.execute("select
avg(pricy),avg(rating),count(restaurant_name) from data where zipcode
between 10006 and 10010")
c = connection.execute("select
avg(pricy),avg(rating),count(restaurant_name) from data where zipcode
between 10011 and 10015")
d = connection.execute("select
avg(pricy),avg(rating),count(restaurant_name) from data where zipcode
between 10016 and 10023")
for data in zip(a,b,c,d):
    print data


# #### Insight -
# + We observe that zipcode 10016 - 10023 has costliest restaurants
with highest pricy rating.
# + Zipcode 10001 - 10005 has cheapest restaurants.
# + Maximum number of restaurants lie in zipcode range 10011 - 10015
and the avg rating rating is also 4.07 which is pretty decent.
```

3. Conclusion

There are number of inferences as mentioned in above notebook code.

They all have been mentioned above in green followed by ##Insight

4. Future Work

We could do a text-based extraction of food items from reviews and find all nearby restaurants serving that particular dish and it’s rating.

In [34]:

```
import py_entitymatching as em #Import megallan entity matching library
import math
import warnings
warnings.filterwarnings('ignore')
```

In [35]:

```
def phone_match(str1,str2):
    if type(str1) is float and type(str2) is float:
        if math.isnan(str1) and math.isnan(str2):
            return True
    elif type(str1) is float:
        if math.isnan(str1):
            return False
    elif type(str2) is float:
        if math.isnan(str2):
            return False
    else:
        stra = ""
        strb = ""
        for ch in str1:
            if ch.isdigit():
                stra += ch
        for ch in str2:
            if ch.isdigit():
                strb += ch
        if stra == strb:
            return True
        else:
            return False
```

In [36]:

```
matching_records = em.read_csv_metadata("before_merging.csv",key="id")
yelp_original = em.read_csv_metadata("yelp_original.csv",key="id")
zomato_original = em.read_csv_metadata("zomato_original.csv",key="id")
```

In [37]:

```
matching_records.columns
```

Out[37]:

```
Index([u'id', u'Unnamed: 1', u'ltable_Name', u'ltable_Phone',
       u'ltable_Zipcode', u'ltable_State', u'ltable_City', u'ltable_Ad
dress',
       u'ltable_Delivery', u'ltable_Takeout', u'ltable_Outdoor_seating
',
       u'rtable_Name', u'rtable_Phone', u'rtable_Zipcode', u'rtable_St
ate',
       u'rtable_City', u'rtable_Address', u'rtable_Delivery',
       u'rtable_Takeout', u'rtable_Outdoor_seating', u'Label', u'predi
cted',
       u'restaurant_name', u'phone', u'zipcode', u'state', u'city', u'
address',
       u'delivery', u'takeout', u'outdoor_seating', u'pricy', u'rating
'],
      dtype='object')
```

In [38]:

```
yelp_original.columns # Original CSV file with added Pricyness column
```

Out[38]:

```
Index([u'id', u'Name', u'Phone', u'Price', u'Zipcode', u'State', u'Cit
y',
       u'Address', u'Has Delivery', u'Has Take-out', u'Has outdoor sea
ting',
       u'Parking'],
      dtype='object')
```

In [39]:

```
zomato_original.columns # Original CSV with added Rating column
```

Out[39]:

```
Index([u'id', u'Name', u'Phone', u'Rating', u'Price', u'Zipcode', u'St
ate',
       u'City', u'Address', u'Delivery', u'Takeout', u'Outdoor', u'Par
king'],
      dtype='object')
```

Schema Merging

In [41]:

```
indexes_to_keep = set()
index = 0
```

```
for index in range(matching_records.shape[0]):
    tuple = matching_records.iloc[index]
    if tuple['predicted'] == 1:

        # Merging the Names -
        # Picking the one that has more length
        if len(tuple['ltable_Name']) > len(tuple['rtable_Name']):
            tuple['restaurant_name'] = tuple['ltable_Name']
        else:
            tuple['restaurant_name'] = tuple['rtable_Name']

        # Merging the Phone no -
        phone1 = tuple['ltable_Phone']
        phone2 = tuple['rtable_Phone']
        if phone_match(phone1, phone2) is True: # When phone numbers are same
            tuple['phone'] = phone1
        else: # Case when phone nos are different. We keep both separated by comma.
            tuple['phone'] = phone1+ "," + phone2

        # Merging the Zipcode -
        # Since blocking was done based on exact match for ZipCode, picking the left
        tuple['zipcode'] = tuple['ltable_Zipcode']

        # Merging the State -
        # Picking the left table attribute
        tuple['state'] = tuple['ltable_State']

        # Merging the City -
        # Picking the left table attribute
        tuple['city'] = tuple['ltable_City']

        # Merging the Address
        # Picking the one that has more length
        if len(tuple['ltable_Address']) > len(tuple['rtable_Address']):
            tuple['address'] = tuple['ltable_Address']
        else:
            tuple['address'] = tuple['rtable_Address']


        tuple['delivery'] = tuple['rtable_Delivery']
        tuple['takeout'] = tuple['rtable_Takeout']
        tuple['outdoor_seating'] = tuple['ltable_Outdoor_seating']

        # For rating
        for ind in range(zomato_original.shape[0]):
            entry = zomato_original.iloc[ind]
            if entry['Name'] == tuple['rtable_Name']:
                tuple['rating'] = entry['Rating']
                break

        # For priciness
        for ind in range(yelp_original.shape[0]):
            entry = yelp_original.iloc[ind]
```

```
            if entry['Name'] == tuple['ltable_Name']:
                tuple['pricy'] = entry['Price']
                break

        # Updating the tuple in predications table
        matching_records.iloc[index] = tuple
        indexes_to_keep.add(index)

    index += 1
```

In [42]:

```
# Print the schema
matching_records.head(1)
```

Out[42]:

	id	Unnamed: 1	ltable_Name	ltable_Phone	ltable_Zipcode	ltable_State	ltable_City	ltable_Addr
0	1	512	McSorley's Old Ale House	(212) 473-9148	10003	NY	New York	15 E 7th

1 rows × 33 columns

In [44]:

```
# Fetch only those rows where predicted = "1" => get correctly matched tuples
sliced = matching_records.take(list(indexes_to_keep))

# Drop columns before merging.
# Dropping old attributes

del sliced['ltable_Name']
del sliced['rtable_Name']
del sliced['ltable_Phone']
del sliced['rtable_Phone']
del sliced['ltable_Zipcode']
del sliced['rtable_Zipcode']
del sliced['ltable_State']
del sliced['rtable_State']
del sliced['ltable_City']
del sliced['rtable_City']
del sliced['ltable_Address']
del sliced['rtable_Address']
del sliced['ltable_Delivery']
del sliced['rtable_Delivery']
del sliced['ltable_Takeout']
del sliced['rtable_Takeout']
del sliced['ltable_Outdoor_seating']
del sliced['rtable_Outdoor_seating']
del sliced['Label'] # Dropping the column'Label'
del sliced['predicted'] # Dropping the column 'predicted

sliced.to_csv("filtered_predictions.csv") # Writing the resultant table to a CSV fi
```

In [1]:

```
import py_entitymatching as em #Import megallan entity matching library
import math
import warnings
warnings.filterwarnings('ignore')

# Pandas has useful data-structure and analysis tool
import pandas as pd
# numpy has a lots of useful math related modules
import numpy as np
# to check execution time
import time
# Helpful function to display intermittent result
from IPython.display import display
```

In [39]:

```
matching_records = em.read_csv_metadata("filtered_predictions.csv",key="id")
yelp_original = em.read_csv_metadata("yelp_original.csv",key="id")
zomato_original = em.read_csv_metadata("zomato_original.csv",key="id")
matching_records.head(1)
```

Out[39]:

	Unnamed: 0	id	Unnamed: 1	restaurant_name	phone	zipcode	state	city	address	delivery	ta
0	0	1	512	McSorley's Old Ale House	(212) 473-9148	10003	NY	New York	15 E 7th Street	0	

Load the dataset

In [3]:

```
#sqlalchemy for sqlite3 use
from sqlalchemy import create_engine
import sqlite3
#create a database where we'll load the dataset from the csv file
engine = create_engine('sqlite:///yelpzomato.db')
connection = engine.connect()
```

In [4]:

```
import string

for data in pd.read_csv("filtered_predictions.csv",
 iterator=True, encoding='utf-8'):
    data.to_sql('data', engine, if_exists='append')
```

impact of delivery facility on pricing and rating

In [5]:

```
a = connection.execute("select AVG(rating) from data where delivery = 0")
b = connection.execute("select AVG(rating) from data where delivery = 1")
for data in zip(a,b):
    print data
```

```
((3.97051282051282,), (4.057500000000001,))
```

Insight -

Takeout facility hasn't affected the rating significantly.

In [6]:

```
a = connection.execute("select AVG(pricy) from data where delivery = 0")
b = connection.execute("select AVG(pricy) from data where delivery = 1")
for data in zip(a,b):
    print data
```

```
((2.3361702127659574,), (1.9,))
```

Insight -

Restaurants that do not offer delivery facility on average are more pricy.

impact of takeout facility on pricing and rating

In [7]:

```
a = connection.execute("select AVG(rating) from data where takeout = 0")
b = connection.execute("select AVG(rating) from data where takeout = 1")
for data in zip(a,b):
    print data
```

```
((3.9120000000000004,), (4.010050251256281,))
```

Insight -

Takeout facility hasn't affected the rating significantly, which means that perhaps people do not care much about outdoor seating

In [8]:

```
a = connection.execute("select AVG(pricy) from data where takeout = 0")
b = connection.execute("select AVG(pricy) from data where takeout = 1")
for data in zip(a,b):
    print data
```

```
((2.763157894736842,), (2.0854271356783918,))
```

Insight -

Restaurants that do not offer takeout facility on average are more pricy.

impact of outdoor seating on pricing and rating

In [9]:

```
a = connection.execute("select AVG(rating) from data where outdoor_seating = 0")
b = connection.execute("select AVG(rating) from data where outdoor_seating = 1")
for data in zip(a,b):
    print data
```

```
((3.999371069182388,), (3.9608695652173904,))
```

Insight -

Outdoor seating hasn't affected the rating significantly, which means that perhaps people do not care much about outdoor seating.

In [30]:

```
a = connection.execute("select AVG(pricy) from data where takeout = 0")
b = connection.execute("select AVG(pricy) from data where takeout = 1")
for data in zip(a,b):
    print data
```

```
((2.763157894736842,), (2.0854271356783918,))
```

Insight -

Restaurants that do not offer outdoor seating on average are more pricy.

impact of delivery and takeout on rating and pricing

In [11]:

```
a = connection.execute("select AVG(rating) from data where delivery = 0 and takeout
b = connection.execute("select AVG(rating) from data where delivery = 0 and takeout
c = connection.execute("select AVG(rating) from data where delivery = 1 and takeout
d = connection.execute("select AVG(rating) from data where delivery = 1 and takeout
for data in zip(a,b,c,d):
    print data
```

```
((3.9120000000000004,), (3.998113207547168,), (None,), (4.057500000000
001,))
```

In [12]:

```
a = connection.execute("select AVG(pricy) from data where delivery = 0 and takeout =
b = connection.execute("select AVG(pricy) from data where delivery = 0 and takeout =
c = connection.execute("select AVG(pricy) from data where delivery = 1 and takeout =
d = connection.execute("select AVG(pricy) from data where delivery = 1 and takeout =
for data in zip(a,b,c,d):
    print data
```

```
((2.763157894736842,), (2.1320754716981134,), (None,), (1.9,))
```

impact of delivery and outdoor seating on rating and pricing

In [13]:

```
a = connection.execute("select AVG(rating) from data where delivery = 0 and outdoor_
b = connection.execute("select AVG(rating) from data where delivery = 0 and outdoor_
c = connection.execute("select AVG(rating) from data where delivery = 1 and outdoor_
d = connection.execute("select AVG(rating) from data where delivery = 1 and outdoor_
for data in zip(a,b,c,d):
    print data
```

```
((3.96439393939394,), (3.978431372549019,), (4.170370370370371,), (3.8
230769230769233,))
```

In [14]:

```
a = connection.execute("select AVG(pricy) from data where delivery = 0 and takeout =
b = connection.execute("select AVG(pricy) from data where delivery = 0 and takeout =
c = connection.execute("select AVG(pricy) from data where delivery = 1 and takeout =
d = connection.execute("select AVG(pricy) from data where delivery = 1 and takeout =
for data in zip(a,b,c,d):
    print data
```

```
((2.763157894736842,), (2.1320754716981134,), (None,), (1.9,))
```

impact of takeout and outdoor seating on rating and pricing

In [15]:

```
a = connection.execute("select AVG(rating) from data where takeout = 0 and outdoor_s
b = connection.execute("select AVG(rating) from data where takeout = 0 and outdoor_s
c = connection.execute("select AVG(rating) from data where takeout = 1 and outdoor_s
d = connection.execute("select AVG(rating) from data where takeout = 1 and outdoor_s
for data in zip(a,b,c,d):
    print data
```

```
((3.9078431372549027,), (3.920833333333333,), (4.042592592592592,), (3
.9714285714285693,))
```

In [16]:

```
a = connection.execute("select AVG(pricy) from data where takeout = 0 and outdoor_se
b = connection.execute("select AVG(pricy) from data where takeout = 0 and outdoor_se
c = connection.execute("select AVG(pricy) from data where takeout = 1 and outdoor_se
d = connection.execute("select AVG(pricy) from data where takeout = 1 and outdoor_se
for data in zip(a,b,c,d):
    print data
```

```
((2.9423076923076925,), (2.375,), (2.1296296296296298,), (2.0329670329
67033,))
```

impact of delivery, takout and outdoor seating on rating and pricing

In [17]:

```
a = connection.execute("select AVG(rating) from data where delivery = 0 and takeout
b = connection.execute("select AVG(rating) from data where delivery = 0 and takeout
c = connection.execute("select AVG(rating) from data where delivery = 0 and takeout
d = connection.execute("select AVG(rating) from data where delivery = 0 and takeout
e = connection.execute("select AVG(rating) from data where delivery = 1 and takeout
f = connection.execute("select AVG(rating) from data where delivery = 1 and takeout
g = connection.execute("select AVG(rating) from data where delivery = 1 and takeout
h = connection.execute("select AVG(rating) from data where delivery = 1 and takeout
for data in zip(a,b,c,d,e,f,g,h):
    print data
```

```
((3.9078431372549027,), (3.920833333333333,), (4.000000000000001,), (3
.9961538461538444,), (None,), (None,), (4.170370370370371,), (3.823076
9230769233,))
```

Interesting Insight -

- Customers prefer restaruants that have delivery and takeout facility and rate them the highest.
- Restaurants that offer all three services overall have received lowest average rating, perhaps since they are cheapest, they do not live upto the standard of the costlier peers in the game.
- Other categories lie in the middle.

In [18]:

```
a = connection.execute("select AVG(pricy) from data where delivery = 0 and takeout =
b = connection.execute("select AVG(pricy) from data where delivery = 0 and takeout =
c = connection.execute("select AVG(pricy) from data where delivery = 0 and takeout =
d = connection.execute("select AVG(pricy) from data where delivery = 0 and takeout =
e = connection.execute("select AVG(pricy) from data where delivery = 1 and takeout =
f = connection.execute("select AVG(pricy) from data where delivery = 1 and takeout =
g = connection.execute("select AVG(pricy) from data where delivery = 1 and takeout =
h = connection.execute("select AVG(pricy) from data where delivery = 1 and takeout =
for data in zip(a,b,c,d,e,f,g,h):
    print data
```

```
((2.9423076923076925,), (2.375,), (2.197530864197531,), (2.06410256410
25643,), (None,), (None,), (1.9259259259259258,), (1.8461538461538463,
))
```

Interesting Insight -

- The priciest restaurants do not offer delivery, takeout and outdoor seating facility. They belong to luxurious class.
- The next category is has the restaurants which are slightly less pricy but just offer outdoor seating facility.
- The trend goes on like this and the restaurants which offer all three facilites are the cheapest.

Data Distribution based on city

In [19]:

```
a = connection.execute("select city, count(city) from data group by city")
for data in a:
    print data
```

```
(u'Brooklyn', 5)
(u'Chicago', 91)
(u'New York', 60)
(u'San Francisco', 85)
(u'Seattle', 34)
```

In [ ]:

```
Brooklyn has very few instances, hence any inference drawn might be inaccurate.
```

impact of city on rating and pricing

In [20]:

```
a = connection.execute("select city, AVG(pricy), AVG(rating) from data group by city
for data in a:
    print data
```

```
(u'Brooklyn', 2.6, 4.0600000000000005)
(u'Chicago', 2.3076923076923075, 4.04175824175824)
(u'New York', 2.5833333333333335, 4.114999999999999)
(u'San Francisco', 2.1058823529411765, 3.8440476190476196)
(u'Seattle', 2.0, 3.9264705882352935)
```

- Shows that Seattle is least pricy but has overall average user rating.
- Restaurants in San Francisco have overall worst rating.
- New York has pricest restaurants with highest user rating.
- Brooklyn - no inference due to lack to adequate number of instances.

Restaurants in Chicago which are most expensive

In [27]:

```
a = connection.execute("select restaurant_name,pricy from data where city = 'Chicago
for data in a:
    print data
```

```
(u'Benny\u2019s Chop House', 4)
(u'Tru', 4)
(u'Joe\u2019s Seafood', 4)
(u'Alinea', 4)
(u'Schwa', 4)
(u'mk The Restaurant', 4)
(u'North Pond', 4)
```

Number of most pricy(4) restaurants in each city

In [29]:

```
a = connection.execute("select count(restaurant_name),city from data where pricy = 4
for data in a:
    print data
```

```
(1, u'Brooklyn')
(7, u'Chicago')
(14, u'New York')
(5, u'San Francisco')
```

- The above result suggests that 'New York' has over all more number of pricy restaurants.
- Based the above result and also the data distribution city wise, we observe that, Seattle is the cheapest of

all followed by San Francisco, then Chicago and New York being the most expensive one.
- Brooklyn has not been considered since there are very few instances of Brooklyn.

Impact of Location on Price and Rating

In [34]:

```
a = connection.execute("select min(zipcode), max(zipcode), city from data group by c
for data in a:
    print data
```

```
(11201, 11231, u'Brooklyn')
(60603, 60661, u'Chicago')
(10001, 10023, u'New York')
(94102, 94133, u'San Francisco')
(98101, 98199, u'Seattle')
```

- The above query gives the range of zipcode for restaurant in records for each city.
- This data can be used for location based analysis.

Location based analysis on New York

In [38]:

```
a = connection.execute("select avg(pricy),avg(rating),count(restaurant_name) from da
b = connection.execute("select avg(pricy),avg(rating),count(restaurant_name) from da
c = connection.execute("select avg(pricy),avg(rating),count(restaurant_name) from da
d = connection.execute("select avg(pricy),avg(rating),count(restaurant_name) from da
for data in zip(a,b,c,d):
    print data
```

```
((2.0625, 4.175, 16), (2.7142857142857144, 4.128571428571428, 7), (2.5
9375, 4.078124999999999, 32), (4.0, 4.14, 5))
```

Insight -

- We observe that zipcode 10016 - 10023 has costliest restaurants with highest pricy rating.
- Zipcode 10001 - 10005 has cheapest restaurants.
- Maximum number of restaurants lie in zipcode range 10011 - 10015 and the avg rating rating is also 4.07 which is pretty decent.

In [22]:

```
#connection.close()
#engine.dispose()
```