Restaurant project

1. Introduction

Indian cities are incredibly diverse collection of restaurants catering to different palettes and appetites. A large part of marketing for a modern restaurant (or any company) is social media, where the number of "likes" that the company can receive will dictate its brand and image to the general public.

For a new business owner (or existing company) to open a new restaurant in major Indian cities, knowing ahead of time the potential social media image they can have would provide an excellent solution to the ever present business problem of uncertainty. In this case the uncertainty is regarding performance of social media presence.

We can mitigate this uncertainty through leveraging data gathered from FourSquare's API, specifically, we are able to scrape "likes" data of different restaurants directly from the API as well as their location and category of cuisine. The question we will try to address is, how accurately can we predict the amount of "likes" a new restaurant opening in this region can expect to have based on the type of cuisine it will serve and which city in India it will open in. (For the purposes of this analysis, we will contain the geographical scope of analysis to three heavily populated cities in Indian, namely Delhi, Mumbai, and Bangalore).

Leveraging this data will solve the problem as it allows the new business owner (or existing company) to make preemptive business decisions regarding opening the restaurant in terms of whether it is feasible to open one in this region and expect good social media presence, what type of cuisine and which city of three would be the best. This project will analyze and model the data via machine learning through comparing both linear and logistic regressions to see which method will yield better predictive capabilities after training and testing.

In [36]:
```
#Let us begin by importing the necessary packages.


import numpy as np

import pandas as pd
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

import json

from geopy.geocoders import Nominatim

import requests
from pandas.io.json import json_normalize


import matplotlib.cm as cm
import matplotlib.colors as colors


import folium

from urllib.request import urlopen
from bs4 import BeautifulSoup


import matplotlib.pyplot as plt
import pylab as pl

from sklearn import linear_model
from sklearn.metrics import jaccard_similarity_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import log_loss
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error, r2_score
import itertools

from geopy.geocoders import Nominatim
import requests
import matplotlib.cm as cm
import matplotlib.colors as colors
import folium

print('Libraries imported.')
```

```
Libraries imported.
```

2. Data

2.1 Data Scraping and Cleaning

In this section we will first retrieve the geographical coordinates of the three cities (Delhi, Mumbai and Bangalore). Then, we will leverage the FourSquare API to obtain URLs that lead to the raw data in JSON form. We will speerately scrape the raw data in these URLs in order to retrieve the following columns: "name", "categories", "latitude", "longitude". and "id" for each city. We can also provide another column ("city") to indicate which city the restaurants are from.

It is important to note that the extracts are not of every restaurant in those cities but rather all of the restaurants within a 2000KM range of the geographical coordinates that geolocator was able to provide. However, the extraction from the FourSquare API actually obtains venue data so it will include venues other than restaurants such as concert halls, stores, libraries etc. As such, this means that the data will need to be further cleaned somewhat manually by removing all of the non-restaurant rows. Once this is complete, we have a shortened by cleaned list to pull "likes" data. The reason the cleaning takes precedence is mainly that pulling the "likes" data is the computing process which takes the longest time in this project so we want to make sure we are not pulling information that will end up being dropped anyways.

The "id" is an important column as it will allow us to further pull the "likes" from the API. We can retreive the "likes" based on the restaurant "id" and then append it to the data frame. Once this is complete, we finally name the dataframe 'raw_dataset' as it is the most complete compiled form before needing any processing for analysis via machine learning.

In [117]:
```
address1 = 'Delhi, India'

geolocator = Nominatim()
location1 = geolocator.geocode(address1)
latitude1 = location1.latitude
longitude1 = location1.longitude
print('The geograpical coordinate of {} are {}, {}.'.format(address1, latitude
1, longitude1))


address2 = 'Mumbai, India'

geolocator = Nominatim()
location2 = geolocator.geocode(address2)
latitude2 = location2.latitude
longitude2 = location2.longitude
print('The geograpical coordinate of {} are {}, {}.'.format(address2, latitude
2, longitude2))

address3 = 'Bangalore, India'

geolocator = Nominatim()
location3 = geolocator.geocode(address3)
latitude3 = location3.latitude
longitude3 = location3.longitude
print('The geograpical coordinate of {} are {}, {}.'.format(address3, latitude
3, longitude3))
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: Deprecati
onWarning: Using Nominatim with the default "geopy/1.21.0" `user_agent` is st
rongly discouraged, as it violates Nominatim's ToS https://operations.osmfoun
dation.org/policies/nominatim/ and may possibly cause 403 and 429 HTTP error
s. Please specify a custom `user_agent` with `Nominatim(user_agent="my-applic
ation")` or by overriding the default `user_agent`: `geopy.geocoders.options.
default_user_agent = "my-application"`. In geopy 2.0 this will become an exce
ption.
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

```
The geograpical coordinate of Delhi, India are 28.6517178, 77.2219388.
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\ipykernel_launcher.py:12: Deprecat
ionWarning: Using Nominatim with the default "geopy/1.21.0" `user_agent` is s
trongly discouraged, as it violates Nominatim's ToS https://operations.osmfou
ndation.org/policies/nominatim/ and may possibly cause 403 and 429 HTTP error
s. Please specify a custom `user_agent` with `Nominatim(user_agent="my-applic
ation")` or by overriding the default `user_agent`: `geopy.geocoders.options.
default_user_agent = "my-application"`. In geopy 2.0 this will become an exce
ption.
  if sys.path[0] == '':
```

```
The geograpical coordinate of Mumbai, India are 18.9387711, 72.8353355.
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\ipykernel_launcher.py:20: Deprecat
ionWarning: Using Nominatim with the default "geopy/1.21.0" `user_agent` is s
trongly discouraged, as it violates Nominatim's ToS https://operations.osmfou
ndation.org/policies/nominatim/ and may possibly cause 403 and 429 HTTP error
s. Please specify a custom `user_agent` with `Nominatim(user_agent="my-applic
ation")` or by overriding the default `user_agent`: `geopy.geocoders.options.
default_user_agent = "my-application"`. In geopy 2.0 this will become an exce
ption.
```

```
The geograpical coordinate of Bangalore, India are 12.9791198, 77.5912997.
```

In [118]:
```
CLIENT_ID = 'WF2HLZPWOXQ0BDJ4DIO4ESR4QBDE5Q3NFM1Y4KQUQEG2WCR2' # Foursquare ID
CLIENT_SECRET = 'YCZKUFTVKWSJZMIU3P4WZA0ZXPE0R3SEKKSYQXCGB3RE1KZC' # Foursquar
e Secret
VERSION = '20180605' # Foursquare API version

print('Your credentails:')
print('CLIENT_ID: ' + CLIENT_ID)
print('CLIENT_SECRET:' + CLIENT_SECRET)


LIMIT = 300 # limit of number of venues returned by Foursquare API
radius = 10000 # define radius

# create URLs
url1 = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secr
et={}&v={}&ll={},{}&radius={}&limit={}'.format(
    CLIENT_ID,
    CLIENT_SECRET,
    VERSION,
    latitude1,
    longitude1,
    radius,
    LIMIT)


# create URLs
url2 = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secr
et={}&v={}&ll={},{}&radius={}&limit={}'.format(
    CLIENT_ID,
    CLIENT_SECRET,
    VERSION,
    latitude2,
    longitude2,
    radius,
    LIMIT)


# create URLs
url3 = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secr
et={}&v={}&ll={},{}&radius={}&limit={}'.format(
    CLIENT_ID,
    CLIENT_SECRET,
    VERSION,
    latitude3,
    longitude3,
    radius,
    LIMIT)

print(url1, url2, url3)
```

```
Your credentails:
CLIENT_ID: WF2HLZPWOXQ0BDJ4DIO4ESR4QBDE5Q3NFM1Y4KQUQEG2WCR2
CLIENT_SECRET:YCZKUFTVKWSJZMIU3P4WZA0ZXPE0R3SEKKSYQXCGB3RE1KZC
https://api.foursquare.com/v2/venues/explore?&client_id=WF2HLZPWOXQ0BDJ4DIO4E
SR4QBDE5Q3NFM1Y4KQUQEG2WCR2&client_secret=YCZKUFTVKWSJZMIU3P4WZA0ZXPE0R3SEKKS
YQXCGB3RE1KZC&v=20180605&ll=28.6517178,77.2219388&radius=10000&limit=300 http
s://api.foursquare.com/v2/venues/explore?&client_id=WF2HLZPWOXQ0BDJ4DIO4ESR4Q
BDE5Q3NFM1Y4KQUQEG2WCR2&client_secret=YCZKUFTVKWSJZMIU3P4WZA0ZXPE0R3SEKKSYQXC
GB3RE1KZC&v=20180605&ll=18.9387711,72.8353355&radius=10000&limit=300 https://
api.foursquare.com/v2/venues/explore?&client_id=WF2HLZPWOXQ0BDJ4DIO4ESR4QBDE5
Q3NFM1Y4KQUQEG2WCR2&client_secret=YCZKUFTVKWSJZMIU3P4WZA0ZXPE0R3SEKKSYQXCGB3R
E1KZC&v=20180605&ll=12.9791198,77.5912997&radius=10000&limit=300
```

In [120]:
```
# scrape the data from the generated URLs

results1 = requests.get(url1).json()
results1

results2 = requests.get(url2).json()
results2

results3 = requests.get(url3).json()
results3

# function that extracts the category of the venue
def get_category_type(row):
    try:
        categories_list = row['categories']
    except:
        categories_list = row['venue.categories']

    if len(categories_list) == 0:
        return None
    else:
        return categories_list[0]['name']


### first city ###

venues1 = results1['response']['groups'][0]['items']
nearby_venues1 = json_normalize(venues1) # flatten JSON

# filter columns
filtered_columns1 = ['venue.name', 'venue.categories', 'venue.location.lat',
                     'venue.location.lng', 'venue.id']
nearby_venues1 =nearby_venues1.loc[:, filtered_columns1]

# filter the category for each row
nearby_venues1['venue.categories'] = nearby_venues1.apply(get_category_type, a
xis=1)

# clean columns
nearby_venues1.columns = [col.split(".")[-1] for col in nearby_venues1.columns
]


### second city ###

venues2 = results2['response']['groups'][0]['items']
nearby_venues2 = json_normalize(venues2) # flatten JSON

# filter columns
filtered_columns2 = ['venue.name', 'venue.categories', 'venue.location.lat',
                     'venue.location.lng', 'venue.id']
nearby_venues2 =nearby_venues2.loc[:, filtered_columns2]

# filter the category for each row
nearby_venues2['venue.categories'] = nearby_venues2.apply(get_category_type, a
xis=1)
```

```
# clean columns
nearby_venues2.columns = [col.split(".")[-1] for col in nearby_venues2.columns
]


### third city ###

venues3 = results3['response']['groups'][0]['items']
nearby_venues3 = json_normalize(venues3) # flatten JSON

# filter columns
filtered_columns3 = ['venue.name', 'venue.categories', 'venue.location.lat',
                     'venue.location.lng', 'venue.id']
nearby_venues3 =nearby_venues3.loc[:, filtered_columns3]

# filter the category for each row
nearby_venues3['venue.categories'] = nearby_venues3.apply(get_category_type, a
xis=1)

# clean columns
nearby_venues3.columns = [col.split(".")[-1] for col in nearby_venues3.columns
]




print('{} venues were returned by Foursquare.'.format(nearby_venues1.shape[0
]))
print('{} venues were returned by Foursquare.'.format(nearby_venues2.shape[0
]))
print('{} venues were returned by Foursquare.'.format(nearby_venues3.shape[0
]))
```

```
100 venues were returned by Foursquare.
100 venues were returned by Foursquare.
100 venues were returned by Foursquare.
```

In [121]:
```
# add locations data to the data sets of each city

nearby_venues1['city'] = 'Delhi'
nearby_venues2['city'] = 'Mumbai'
nearby_venues3['city'] = 'Bangalore'
```

In [122]:
```
# combine the three cities into one data set

nearby_venues = nearby_venues1.copy()
nearby_venues = nearby_venues.append(nearby_venues2)
nearby_venues = nearby_venues.append(nearby_venues3)
```

In [123]:
```
# check list and manually remove all non-restaurant data

nearby_venues['categories'].unique()

removal_list = ['Concert Hall', 'Opera House', 'Dance Studio',
                'Performing Arts Venue', 'Art Museum', 'Park',
                'Massage Studio', 'Music Venue', 'Bookstore', 'Clothing Store'
,
                'Boutique', 'Furniture/Home Store', 'Jazz Club',
                'Theater', 'Optical Shop', "Men's Store", 'Rock Club',
                'Gym / Fitness Center', 'Wine Shop', 'Indie Movie Theater',
                'Chocolate Shop', 'Dessert Shop', 'Recreation Center',
                'Plaza', 'Hotel', 'Luggage Store', 'Farmers Market', 'Gym',
                'Jewelry Store', 'Furniture / Home Store', 'Butcher',
                'Bakery', 'Marijuana Dispensary', 'Ice Cream Shop',
                'Comic Shop', 'Bagel Shop', 'Spa', 'Liquor Store', 'Bike Shop'
,
                'Yoga Studio', 'Pedestrian Plaza', 'Candy Store',
                'Park', 'Bookstore', 'Candy Store',  'Jazz Club', 'Art Galler
y',
                 'Supermarket', 'Museum', 'Boutique', 'Plaza', 'Building', 'Ba
kery',
                 'Historic Site', 'Ice Cream Shop', ' Concert Hall', 'Pharmac
y',
                 'Market', 'Movie Theater', 'Performing Arts Venue', 'Music Ve
nue',
                 'Theater', 'Art Museum', 'Cheese Shop', 'Opera House',
                 'Pedestrian Plaza', 'School', 'Gift Shop', 'Athletics & Sport
s',
                 'Shoe Repair', 'General Entertainment', 'Stationery Store',
                 'Toy / Game Store', 'Brewery', 'Hotel', 'Theater', 'Music Ven
ue', 'Business Service',
                 'Donut Shop', 'Liquor Store', 'Beer Store',
                 'Lounge', 'Plaza', 'Health Food Store', 'Concert Hall',
                 'Lingerie Store', 'Gym', 'Mobile Phone Shop',
                 'Chocolate Shop', 'Ice Cream Shop', 'Hostel', 'Convenience St
ore',
                 'Park', 'Farmers Market', 'Cosmetics Shop', 'Piano Bar',
                 'Nightclub', 'Massage Studio', 'Comedy Club', 'Concert Hall']

nearby_venues = nearby_venues[~nearby_venues['categories'].isin(removal_list)]

nearby_venues['categories'].unique()
```

Out[123]:
```
array(['Indian Restaurant', 'Restaurant', 'Snack Place',
       'South Indian Restaurant', 'Smoke Shop', 'Tibetan Restaurant',
       'Food Truck', 'Molecular Gastronomy Restaurant', 'Mosque',
       'Bistro', 'Bar', 'Deli / Bodega', 'Coffee Shop', 'Café',
       'Spiritual Center', 'Italian Restaurant', 'Sculpture Garden',
       'North Indian Restaurant', 'Monument / Landmark', 'Arcade',
       'Food & Drink Shop', 'Irani Cafe', 'Golf Course',
       'Fast Food Restaurant', 'Playground', 'Mediterranean Restaurant',
       'French Restaurant', 'Karnataka Restaurant', 'Sandwich Place',
       'Northeast Indian Restaurant', 'BBQ Joint', 'Hindu Temple',
       'Cocktail Bar', 'Cricket Ground', 'Chinese Restaurant',
       'Parsi Restaurant', 'Scenic Lookout', 'Seafood Restaurant',
       'Beach', 'Diner', 'Pub', 'History Museum', 'Pizza Place',
       'Breakfast Spot', 'Asian Restaurant', 'Japanese Restaurant',
       'New American Restaurant', 'Thai Restaurant', 'Flea Market',
       'Middle Eastern Restaurant', 'Club House', 'Steakhouse', 'Stadium',
       'Shopping Mall', 'Hotel Bar', 'Cupcake Shop', 'Tea Room',
       'American Restaurant', 'Burger Joint', 'Sushi Restaurant',
       'Afghan Restaurant', 'Department Store', 'Botanical Garden',
       'Mexican Restaurant', 'Multiplex', 'Bowling Alley', 'Juice Bar',
       'Gaming Cafe', 'Financial or Legal Service', 'German Restaurant',
       'Pakistani Restaurant', 'Creperie'], dtype=object)
```

In [ ]:
```
# set up to pull the likes from the API based on venue ID

url_list = []
like_list = []
json_list = []

for i in list(nearby_venues.id):
    venue_url = 'https://api.foursquare.com/v2/venues/{}/likes?client_id={}&cl
ient_secret={}&v={}'.format(i, CLIENT_ID, CLIENT_SECRET, VERSION)
    url_list.append(venue_url)
for link in url_list:
    result = requests.get(link).json()
    likes = result['response']['likes']['count']
    like_list.append(likes)
print(like_list)


nearby_venues['likes'] = like_list
```

In [414]:
```
# Now let us rename this raw dataset

raw_dataset = nearby_venues
raw_dataset.head()
```

Out[414]:

	name	categories	lat	lng	id	city
5	Bengali Market | बंगाली मार्केट | বাংলা বাজার	Indian Restaurant	28.629498	77.232020	4ba05f35f964a520d16a37e3	Delhi
7	Tamra	Restaurant	28.620543	77.218174	54dc85c7498ef8f9ab9b3c08	Delhi
8	Amritsari Lassi Wala	Snack Place	28.657325	77.224138	5662936e498e19a9801a663f	Delhi
10	Sagar Ratna	Indian Restaurant	28.635487	77.220650	4cb876d7f50e224bd2d6e6fb	Delhi
11	HOTEL SARAVANA BHAVAN	South Indian Restaurant	28.632319	77.216445	519ba450498eb0c559152d94	Delhi

In [415]:
```
address = 'Delhi, IN'
geolocator = Nominatim(user_agent="foursquare_agent")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinate of Delhi is {}, {}.'.format(latitude,longitude))

# create map of Toranto using latitude and longitude values
map_delhi = folium.Map(location=[latitude, longitude], zoom_start=13)

# add markers to map
for lat, lng, borough, neighborhood in zip(raw_dataset['lat'], raw_dataset['lng'], raw_dataset['name'], raw_dataset['categories']):
    label = '{}, {}'.format(neighborhood, borough)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(map_delhi)
map_delhi
```

```
The geograpical coordinate of Delhi is 28.6517178, 77.2219388.
```

Out[415]:

In [416]:
```
address = 'Mumbai, IN'
geolocator = Nominatim(user_agent="foursquare_agent")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinate of Mumbai is {}, {}.'.format(latitude,longit
ude))

# create map of Toranto using latitude and longitude values
map_mumbai = folium.Map(location=[latitude, longitude], zoom_start=15)

# add markers to map
for lat, lng, borough, neighborhood in zip(raw_dataset['lat'], raw_dataset['ln
g'], raw_dataset['name'], raw_dataset['categories']):
    label = '{}, {}'.format(neighborhood, borough)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(map_mumbai)
map_mumbai
```

```
The geograpical coordinate of Mumbai is 18.9387711, 72.8353355.
```

Out[416]:

In [417]:
```
address = 'Bangalore, IN'
geolocator = Nominatim(user_agent="foursquare_agent")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinate of Bangalore is {}, {}.'.format(latitude,lon
gitude))

# create map of Toranto using latitude and longitude values
map_Bangalore = folium.Map(location=[latitude, longitude], zoom_start=15)

# add markers to map
for lat, lng, borough, neighborhood in zip(raw_dataset['lat'], raw_dataset['ln
g'], raw_dataset['name'], raw_dataset['categories']):
    label = '{}, {}'.format(neighborhood, borough)
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#3186cc',
        fill_opacity=0.7,
        parse_html=False).add_to(map_Bangalore)
map_Bangalore
```

```
The geograpical coordinate of Bangalore is 12.9791198, 77.5912997.
```

Out[417]:

2.2 Data Preparation

The data still needs some more processing before it is suitable for model training and testing. Mainly, the "categories" column contains too many different types of cuisines to allow a model to yield any meaningful results. However, the different types of natural cuisines have natural groupings based on conventionally accepted cultural groupings of cuisine. Broadly speaking, all of the different types of cuisine could be reclassified as European, Latin American, Asian, North American, drinking establishments (bars), or casual establishments such as coffee shops or ice cream parlours. We can implement manual classification as there really aren't that many different types of cuisines.

As this project will compare both linear and logistic regression, it makes sense to have "likes" as both a continuous and categorical (but ordinal) variable. In the case of turning into a categorical variable, we can bin the data based on percentiles and classify them into these ordinal percentile categories. I tried different ways of binning but in the end, splitting the sample into three different bins proved to yield the best classification results from a prediction standpoint.

As the last stage of data preparation, it is important to note that the regressors are categorical variables (3 different cities and 6 different categories of cusines). Hence, they require dummy variable encoding for meaningful analysis. We can accomplish this via one-hot encoding.

In [13]:
```
# inspecting the raw dataset shows that there may be too many different types
 of cuisines
raw_dataset['categories'].unique()
```

Out[13]:
```
array(['Snack Place', 'Indian Restaurant', 'Food & Drink Shop', 'Mosque',
       'Tibetan Restaurant', 'Restaurant', 'Korean Restaurant',
       'Hardware Store', 'Paper / Office Supplies Store', 'Bar', 'Café',
       'Fast Food Restaurant', 'Food', 'Indian Chinese Restaurant',
       'Motel', 'Pizza Place', 'Flea Market', 'Breakfast Spot',
       'Coffee Shop', 'Sandwich Place', 'Light Rail Station',
       'Miscellaneous Shop', 'Road', 'Falafel Restaurant',
       'Chinese Restaurant', 'Parsi Restaurant', 'Cricket Ground',
       'Seafood Restaurant', 'Scenic Lookout', 'Beach',
       'Asian Restaurant', 'History Museum', 'Italian Restaurant',
       'Japanese Restaurant', 'Music Store', 'Train Station', 'Pub',
       'Middle Eastern Restaurant', 'Gastropub', 'Diner',
       'New American Restaurant', 'Multiplex',
       'College Academic Building', 'Stadium', 'Cocktail Bar',
       'Monument / Landmark', 'Hockey Arena', 'Mediterranean Restaurant',
       'Chaat Place', 'BBQ Joint', 'Tea Room', 'Shopping Mall',
       'French Restaurant', 'Deli / Bodega', 'American Restaurant',
       'Cupcake Shop', 'Racetrack', 'Burger Joint', 'Sushi Restaurant',
       'Steakhouse', 'Afghan Restaurant', 'Bubble Tea Shop',
       'Golf Course', 'Fried Chicken Joint', 'Hotel Bar',
       'Vietnamese Restaurant', 'Bed & Breakfast', 'Arcade', 'Wine Bar',
       'Electronics Store', 'Vegetarian / Vegan Restaurant',
       'South Indian Restaurant', 'Karnataka Restaurant'], dtype=object)
```

In [138]:
```
euro = ['French Restaurant', 'Scandinavian Restaurant', 'Souvlaki Shop',
       'Mediterranean Restaurant', 'Italian Restaurant', 'Pizza Place']

latino = ['Mexican Restaurant', 'Latin American Restaurant',
          'Brazilian Restaurant', 'Taco Place', "Fast Food Restaurant",
          "Breakfast Spot","Tea Room","Bubble Tea Shop", 'Cupcake Shop']

bar = ['Beer Bar', 'Cocktail Bar', 'Tiki Bar', 'Wine Bar', 'Hotel Bar',
       'Beer Garden', 'Speakeasy', 'Brewery', 'Pub', 'Bar', 'Gastropub',
       'Hookah Bar']

asian = ['Ramen Restaurant', 'Sushi Restaurant', 'Vietnamese Restaurant',
         'Thai Restaurant', 'Poke Place', 'Indian Restaurant',
         'Japanese Curry Restaurant', 'Japanese Restaurant',
         'Indonesian Restaurant', 'Udon Restaurant', 'Noodle House',
         'Falafel Restaurant', 'Filipino Restaurant', 'Turkish Restaurant',
         'Yoshoku Restaurant','snack place','Shyam Sweet','South Indian Restau
rant',
        'Karnataka Restaurant', 'Vegetarian / Vegan Restaurant', 'Indian Resta
urant',
        'Tibetan Restaurant', 'Restaurant', 'Korean Restaurant']

casual = ['Coffee Shop', 'Café', 'Sandwich Place', 'Food Truck',
          'Juice Bar', 'Frozen Yogurt Shop', 'Deli / Bodega', 'Dessert Shop',
          'Hot Dog Joint', 'Burger Joint', 'Breakfast Spot',
          'Fondue Restaurant',"Afghan Restaurant", 'Food & Drink Shop']

american = ['Southern / Soul Food Restaurant', 'Food & Drink Shop',
            'Restaurant', 'American Restaurant', 'BBQ Joint',
            'Theme Restaurant', 'New American Restaurant',
            'Vegetarian / Vegan Restaurant', 'Bed & Breakfast','Seafood Restau
rant']

def conditions(s):
    if s['categories'] in euro:
        return 'euro'
    if s['categories'] in latino:
        return 'latino'
    if s['categories'] in asian:
        return 'asian'
    if s['categories'] in casual:
        return 'casual'
    if s['categories'] in american:
        return 'american'
    if s['categories'] in bar:
        return 'bar'


raw_dataset['categories_classified']=raw_dataset.apply(conditions, axis=1)
raw_dataset.head()
```

Out[138]:

	name	categories	lat	lng	id	city	likes	categories
0	Amritsari Lassi Wala	Snack Place	28.657325	77.224138	5662936e498e19a9801a663f	Delhi	6	
2	Kake Di Hatti | काके दी हट्टी	Indian Restaurant	28.658050	77.223377	4d9d759348b6224b70c2249f	Delhi	34	
5	Spice Market	Food & Drink Shop	28.657287	77.222595	5280a63211d26b82c4ba65c7	Delhi	36	
7	Karim's | करीम | كريم (Karim's)	Indian Restaurant	28.649498	77.233691	4b42e3c7f964a520b5da25e3	Delhi	316	
8	Jama Masjid | जामा मस्जिद | جامع مسجد (Jama Ma...	Mosque	28.650136	77.233541	4b529b68f964a520b48327e3	Delhi	177	

In [139]:
```
# double check to make sure categories_classified has been created correctly
raw_dataset['categories_classified'].value_counts()
```

Out[139]:
```
asian       57
casual      24
latino      17
bar         15
euro        12
american     8
Name: categories_classified, dtype: int64
```

In [140]:
```
# classify the likes into different ranking levels
# lets first see where to bin the data
# we can try different ways of binning the data, I find it yields substantiall
y different results

print(np.percentile(raw_dataset['likes'], 66))


print(np.percentile(raw_dataset['likes'], 76))
```

```
35.52000000000001
63.96000000000015
```

In [351]:
```
# create a function to bin for us
def rankings(s):
    if s['likes']<=15:
        return 1
    if s['likes']<=37:
        return 2
    if s['likes']>37:
        return 3

raw_dataset['ranking']=raw_dataset.apply(rankings, axis=1)
raw_dataset.head()
```

Out[351]:

	name	categories	lat	lng	id	city	likes	categories
0	Amritsari Lassi Wala	Snack Place	28.657325	77.224138	5662936e498e19a9801a663f	Delhi	6	
2	Kake Di Hatti | काके दी हट्टी	Indian Restaurant	28.658050	77.223377	4d9d759348b6224b70c2249f	Delhi	34	
5	Spice Market	Food & Drink Shop	28.657287	77.222595	5280a63211d26b82c4ba65c7	Delhi	36	
7	Karim's | करीम | كريم (Karim's)	Indian Restaurant	28.649498	77.233691	4b42e3c7f964a520b5da25e3	Delhi	316	
8	Jama Masjid | जामा मस्जिद | جامع مسجد (Jama Ma...	Mosque	28.650136	77.233541	4b529b68f964a520b48327e3	Delhi	177	

In [352]:
```
raw_dataset['ranking'].value_counts()
```

Out[352]:
```
1    75
3    60
2    52
Name: ranking, dtype: int64
```

In [353]:
```
# create dummies for linear regression modelling

# one hot encoding
reg_dataset = pd.get_dummies(raw_dataset[['categories_classified',
                                          'city',]],
                             prefix="",
                             prefix_sep="")

# add name, ranking, and likes columns back to dataframe
reg_dataset['ranking'] = raw_dataset['ranking']
reg_dataset['likes'] = raw_dataset['likes']
reg_dataset['name'] = raw_dataset['name']

# move name column to the first column
reg_columns = [reg_dataset.columns[-1]] + list(reg_dataset.columns[:-1])
reg_dataset = reg_dataset[reg_columns]


reg_dataset.head()
```

Out[353]:

	name	american	asian	bar	casual	euro	latino	Bangalore	Delhi	Mumbai	ranking	like
0	Amritsari Lassi Wala	0	0	0	0	0	0	0	1	0	1	
2	Kake Di Hatti | काके दी हट्टी	0	1	0	0	0	0	0	1	0	2	3
5	Spice Market	0	0	0	1	0	0	0	1	0	2	3
7	Karim's | करीम | كريم (Karim's)	0	1	0	0	0	0	0	1	0	3	31
8	Jama Masjid | जामा मस्जिद | جامع مسجد (Jama Ma...	0	0	0	0	0	0	0	1	0	3	17

3. Methodology

This project will utilize both linear and logistic regression machine learning methods to train and test the data. Namely, linear regression will be used in an attempt to predict the number of "likes" a new restaurant in this region will have. We will utilize the Sci-Kit Learn Package to run the model.

We can also utilize logisitc regression as a classification method rather than direct prediction of the number of likes. Since the number of "likes" can be binned into different categories based on different percentile bins, it is also potentiallly possible to see which range of "likes" a new restaurant in this region will have.

Since the "likes" are binned into multiple (more than 2) categories, the type of logistic regression will be multinomial. Additionally, although the ranges are indeed discrete categories, they are also ordinal in nature. Therefore the logistic regression will need to be specified as being both multinomial and ordinal. This can be done through the Sci-Kit Learn Package as well.

4. Results

4.1 Linear Regression Results

A linear regression model was trained on a random subsample of 80% of the sample and then tested on the other 20%. To see if this is a reasonable model, the residual sum of squares score and variance score were both calculated. Given the low variance score, this is probably not a valid/good way of modelling the data. Therefore, we move on to logistic regression.

In [407]:
```
# Multiple Linear Regression

msk = np.random.rand(len(reg_dataset)) < 0.8
train = reg_dataset[msk]
test = reg_dataset[~msk]

regr = linear_model.LinearRegression()
x = np.asanyarray(train[['american', 'asian', 'bar', 'casual',
                         'euro',  'Mumbai',
                         'Bangalore', 'Delhi']])
y = np.asanyarray(train[['likes']])
regr.fit (x, y)
# The coefficients
print ('Coefficients: ', regr.coef_)
```

```
Coefficients:  [[-32.02767096 -29.71836321 -13.90550675  43.86342662  25.4386
1746
   16.14704287  10.59350449 -26.74054736]]
```

In [408]:
```
# Multiple Linear Regression Prediction Capabilities

y_hat= regr.predict(test[['american', 'asian', 'bar', 'casual',
                          'euro','Mumbai',
                          'Bangalore', 'Delhi']])
x = np.asanyarray(test[['american', 'asian', 'bar', 'casual',
                          'euro', 'Mumbai',
                          'Bangalore', 'Delhi']])
y = np.asanyarray(test[['likes']])
print("Residual sum of squares: %.2f"
      % np.mean((y_hat - y) ** 2))

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % regr.score(x, y))
```

```
Residual sum of squares: 22502.75
Variance score: -0.01
```

4.2 Logistic Regression Results

A multinomial ordinal logisitc regression model was trained on a random subsample of 80% of the sample and then tested on the other 20%. To see if this is a reasonable model, its jaccard similarity score and log-loss were calculated. Although this is not a perfect prediction, a similarity of 62% between the training set and test set is a reasonable result. The classification report is also printed later on below.

Given the modestly accurate ability of this model, we can also run the model on the full dataset. The coefficients show that opening a restaurant in Delhi, opening a bar, or serving cuisine that is Latino or casual in nature, are associated positevely with "likes."

In [409]:
```
# Multinomial Ordinal Logistic Regression

x_train = np.asanyarray(train[['american', 'asian', 'bar', 'casual',
                          'euro','latino', 'Mumbai',
                          'Bangalore', 'Delhi']])
y_train = np.asanyarray(train['ranking'])

x_test = np.asanyarray(test[['american', 'asian', 'bar', 'casual',
                          'euro', 'latino', 'Mumbai',
                          'Bangalore', 'Delhi']])
y_test = np.asanyarray(test['ranking'])

mul_ordinal = linear_model.LogisticRegression(multi_class='multinomial', solve
r='newton-cg',fit_intercept=True).fit(x_train,
                                                                    y_train)
mul_ordinal
coef = mul_ordinal.coef_[0]
print (coef)
```

```
[-0.23409404  0.27837899 -0.73521561 -0.3322533  -0.41291794  0.15150594
 -0.4894959  -0.01753665  0.50703976]
```

In [410]:
```
# Multinomial Ordinal Logistic Regression Prediction Capabilities

yhat = mul_ordinal.predict(x_test)
#yhat
yhat_prob = mul_ordinal.predict_proba(x_test)
#yhat_prob
jaccard_similarity_score(y_test, yhat)
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:
635: DeprecationWarning: jaccard_similarity_score has been deprecated and rep
laced with jaccard_score. It will be removed in version 0.23. This implementa
tion has surprising behavior for binary and multiclass classification tasks.
  'and multiclass classification tasks.', DeprecationWarning)
```

Out[410]:
```
0.6176470588235294
```

In [411]:
```
log_loss(y_test, yhat_prob)
```

Out[411]:
```
0.9489002607869752
```

In [412]:
```
# Exploration of Coefficient Magnitudes of Full Dataset

x_all = np.asanyarray(reg_dataset[['american', 'asian', 'bar', 'casual',
                                   'euro', 'Mumbai',
                             'Bangalore', 'Delhi']])
y_all = np.asanyarray(reg_dataset['ranking'])



LR = linear_model.LogisticRegression(multi_class='multinomial',
                                             solver='newton-cg',
                                             fit_intercept=True).fit(x_all,
                                                                     y_all)

LR

coef = LR.coef_[0]
print (coef)
```

```
[-0.5277577   0.27254126 -0.92976511 -0.4127115  -0.30271641 -0.54950532
 -0.02691445  0.5764283 ]
```

In [413]:
```
print (classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

           1       0.67      0.91      0.77        11
           2       0.50      0.40      0.44        10
           3       0.64      0.54      0.58        13

    accuracy                           0.62        34
   macro avg       0.60      0.62      0.60        34
weighted avg       0.61      0.62      0.60        34
```

5. Discussion

The first thing to note is that given the data, logistic regression presents a better fit for the data over linear regression. Using logistic regression we were able to obtain a Jaccard Similarity Score of 61.76%, which although not perfect, but more reasonable than the low variance score obtained from the linear regression. As stated before, please note that for the purposes of this project, we are assumming that likes are a good proxy for how well a new restaurant will do in terms of brand, image and by extension how well the restaurant will perform business-wise. Whether or not these assumptions hold up in a real-life scenario is up for discussion, but this project does contain limitations in scope due to the amount of data that can be fetched from the FourSquare API.

As such, to obtain insights into this data, we can proceed with breaking down the results of the logistic regression model. The results showed that the precision score for classifying whether the new restaurant would fall into classes 1, 2, or 3 (lowest, medium, or highest percentile of likes) were 70%, 54%, and 69%. Therefore, the model is better at predicting if a restaurant will fall into the best or worst percentile of likes. This is good as we are mostly concerned with whether the restuarant will perform well or not so the high accuracy of predictions for the two extremum is a welcome feature. This allows us to fairly accurately predict the general performance of the business opportunity. Different binning(Created 5 and 10 bins) methods for the classes were attempted, but the use of 3 bins by far yielded the best Jaccard Similarity Score.

Additionally, not only are we attempting to predict the general business performance but also pull insights to inform on business strategy. In this case strategy insight can be gleamed from the coefficient values from running the logistic regressin on the full dataset. As such, we can see that opening a restaurant in Delhi, opening a bar, or serving cuisine that is Latino or casual in nature, are associated positevely with "like." This suggests that the business opportunity should be opening a restaurant in either Mumbai or Bangalore, with a cuisine that is American or bars in nature would be the best approach for maximizing likes.

6. Conclusion

In conclusion, after analyzing restaurant "likes" in India from 300 restaurants, we can conclude that the approach to best take in regards to maximizing business performance (as measured by "likes") is to open a restaurant that is either Latino or casual and that opening the venue in either Delhi or Bangalore would be the best approach. Additionally, the predictive capabilities of the logistic regression prediction model are most accurate for classifying whether a restaurant will fall in either the best or worst classes when the data is binned into 3 classes.