Yelp Recommendation Models

August 30, 2019

1 RECOMMENDING RESTAURANTS

Table of Contents 1. Section **??**

2. Section **??**

[3]:
```
import numpy as np
import pandas as pd
import json
import csv
import os
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import nltk
from nltk.corpus import stopwords
import re
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
#conda install conda-forge wordcloud
from wordcloud import WordCloud
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
#can't install new tensorflow cuz of macos so use an older version
# import tensorflow as tf
# from tensorflow import keras
import geopandas
import descartes
import requests
import urllib
import gc
from shapely.geometry import Point
# from arcgis.gis import GIS
```

1.1 SET UP CODE

[4]:
```
%%time
# Reading the business dataset
# columns include address, attributes, business_id,
# categories, city, hours, is_open, latitude, longitude,
# name, postal_code, review_count, stars, state

business = pd.read_csv('yelp_business.csv')
checkin = pd.read_csv('yelp_checkin.csv')
user = pd.read_csv('yelp_user.csv')
review = pd.read_csv('yelp_review.csv')
```

```
CPU times: user 1min 6s, sys: 26.1 s, total: 1min 32s
Wall time: 1min 46s
```

[ ]:
```
%%time
# # The total number of businesses: 174567
# print(business.shape[0])
# # The number of businesses for each state
# #AZ = 52214, NV = 33086, ON = 30208
# business['state'].value_counts()

# Found data on outside of US. We should exclude international businesses.

stateinitials = ['AL', 'AK', 'AZ', 'AR', 'CA',
                 'CO', 'CT', 'DE', 'FL', 'GA', 'HI',
                 'ID', 'IL', 'IN', 'IA', 'KS', 'KY', 'LA',
                 'ME', 'MD', 'MA', 'MI', 'MN', 'MS',
                 'MO', 'MT', 'NE', 'NV', 'NH', 'NJ', 'NM',
                 'NY', 'NC', 'ND', 'OH', 'OK', 'OR', 'PA',
                 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VT',
                 'VA', 'WA', 'WV', 'WI', 'WY']
# Filter for only USA LOCATIONS
usbusiness = business[business['state'].isin(stateinitials)]
# Filter for Food and Restaurants Including Categories

foodbusiness = usbusiness[usbusiness['categories'].apply(lambda x: 'Food' in␣
 ↪str(x) or 'Restaurant' in str(x))]

#number of businesses in the food section
#AZ = 13826, NV = 9263, OH = 6031, NC = 4969
# foodbusiness['state'].value_counts()

# the food section includes supermarkets and nonrestaurant food related␣
 ↪businesses
nonrestaurantlist = ['Shopping', 'Shopping,',
```

```
                    'Services', 'Groceries', 'Event Planning',␣
 ↪'Convenience','Convenience Stores',
                   'Gas Station', 'Gas', 'Grocery', 'Station', 'Banks,']
# filter out sole food serving places
foodbusiness = foodbusiness[foodbusiness['categories'].apply(
    lambda x: not any(s in nonrestaurantlist for s in str(x).split(';')))]

#change categories into a list not a string
foodbusiness['categorieslist'] = foodbusiness['categories'].apply(lambda x:␣
 ↪str(x).split(';'))
# # number of food businesses for each state
# # AZ 12331, NV 8399, OH = 5535
# foodbusiness['state'].value_counts()

#selecting only AR businesses
arizonafoodbusiness = foodbusiness[foodbusiness['state'] == 'AZ']
#selecting only NV businesses
nevadafoodbusiness = foodbusiness[foodbusiness['state'] == 'NV']

restaurantgroups = ['Afghan',
'African', 'Senegalese', 'South African', 'American (New)', 'American␣
 ↪(Traditional)',
'Arabian','Argentine','Armenian','Asian Fusion','Australian','Austrian',
'Bangladeshi','Barbeque','Basque','Belgian','Brasseries','Brazilian',
'Breakfast & Brunch','Pancakes','British,Buffets',
'Bulgarian','Burgers','Burmese','Cafes','Themed Cafes','Cafeteria','Cajun/
 ↪Creole','Cambodian','Caribbean','Dominican','Haitian','Puerto␣
 ↪Rican','Trinidadian','Catalan','Cheesesteaks','Chicken Shop','Chicken␣
 ↪Wings','Chinese',
'Cantonese','Dim Sum','Hainan','Shanghainese','Szechuan','Comfort Food',
'Creperies','Cuban','Czech','Delis','Diners','Dinner␣
 ↪Theater','Eritrean','Ethiopian','Fast Food','Filipino','Fish &␣
 ↪Chips','Fondue','Food Court','Food␣
 ↪Stands','French','Mauritius','Reunion','Game Meat','Gastropubs','Georgian',
'German','Gluten-Free','Greek','Guamanian','Halal','Hawaiian','Himalayan/
 ↪Nepalese',
'Honduran','Hong Kong Style Cafe','Hot Dogs','Hot␣
 ↪Pot','Hungarian','Iberian','Indian','Indonesian','Irish','Italian','Calabrian','Sardinian','S
 ↪Belt Sushi','Izakaya','Japanese Curry','Ramen','Teppanyaki',
'Kebab','Korean','Kosher','Laotian','Latin␣
 ↪American','Colombian','Salvadoran','Venezuelan','Live/Raw␣
 ↪Food','Malaysian','Mediterranean','Falafel','Mexican',
'Tacos','Middle Eastern','Egyptian','Lebanese','Modern European','Mongolian',
```

```
'Moroccan','New Mexican Cuisine','Nicaraguan','Noodles','Pakistani','Pan␣
 ↪Asian','Persian/Iranian','Peruvian','Pizza','Polish','Polynesian','Pop-Up␣
 ↪Restaurants','Portuguese','Poutineries','Russian','Salad','Sandwiches','Scandinavian','Scotti
 ↪Food','Soup','Southern','Spanish','Sri Lankan','Steakhouses','Supper␣
 ↪Clubs','Sushi Bars','Syrian','Taiwanese','Tapas Bars',
'Tapas/Small␣
 ↪Plates','Tex-Mex','Thai','Turkish','Ukrainian','Uzbek','Vegan','Vegetarian','Vietnamese','Waf
'Wraps']

arizonafoodbusiness['maincuisine'] = arizonafoodbusiness['categorieslist'].apply(
    lambda x: [a for a in x if a in restaurantgroups]).apply(
    lambda h: h[0] if len(h) != 0 else np.nan)

# filter to include only the business id of the food business
foodcheckin = checkin[checkin['business_id'].isin(foodbusiness['business_id'])]
# filter only arizona food businesses
arizonafoodcheckin = checkin[checkin['business_id'].
 ↪isin(arizonafoodbusiness['business_id'])]
# sum of checkins per business for food business
sumfoodcheckin = foodcheckin.groupby('business_id').agg(
    {'checkins': 'sum'}).reset_index()
sumfoodcheckin = dict(zip(sumfoodcheckin['business_id'],
                          sumfoodcheckin['checkins']))
#adding it to the az and food df
arizonafoodbusiness['sumcheckin'] = arizonafoodbusiness['business_id'].
 ↪map(sumfoodcheckin)
foodbusiness['sumcheckin']= foodbusiness['business_id'].map(sumfoodcheckin)

# # change the date into year, day
# review['dateobject'] = review['date'].apply(lambda x: datetime.datetime.
 ↪strptime(x, '%Y-%m-%d'))
# review['year' ] = review['dateobject'].apply(lambda h: h.year)
# review['day'] = review['dateobject'].apply(lambda h: h.weekday())

# #include reviews of only food businesses in our study
foodreviews = review[review['business_id'].isin(foodbusiness['business_id'])]
#include reviews of only food businesses in Arizona
arizonafoodreviews = review[review['business_id'].
 ↪isin(arizonafoodbusiness['business_id'])]

# include reviews of only food businesses in Nevada
nevadafoodreviews = review[review['business_id'].
 ↪isin(nevadafoodbusiness['business_id'])]

# food reviews from users living in Arizona
userarizona = arizonafoodreviews['user_id'].unique()
```

```
#business id to name
businessidtoname = dict(zip(arizonafoodbusiness['business_id'],␣
 ↪arizonafoodbusiness['name']))
#business name to lon
businessidtolong = dict(zip(arizonafoodbusiness['business_id'],␣
 ↪arizonafoodbusiness['longitude']))
#business name to lat
businessidtolat = dict(zip(arizonafoodbusiness['business_id'],␣
 ↪arizonafoodbusiness['latitude']))

# add business name to AZ food reviews
arizonafoodreviews['business_name'] = arizonafoodreviews['business_id'].
 ↪map(businessidtoname)

#add lowername to the AZ food businesses
arizonafoodbusiness['lowername'] = arizonafoodbusiness['name'].apply(lambda x: x.
 ↪replace("'", "").lower())

#add set of lon and lat to AZ food reviews
arizonafoodreviews['lon'] = arizonafoodreviews['business_id'].
 ↪map(businessidtolong)
arizonafoodreviews['lat'] = arizonafoodreviews['business_id'].
 ↪map(businessidtolat)

# #min year and max year review for business id
# minyear = arizonafoodreviews.groupby('business_id').agg(
#     {'year': np.min}).reset_index()
# maxyear = arizonafoodreviews.groupby('business_id').agg(
#     {'year': np.max}).reset_index()
# minmaxyears = minyear.merge(maxyear, on = 'business_id', how = 'inner')
# #dictionary of business's min and max year of reviews
# minyear = dict(zip(minmaxyears['business_id'],
#                    minmaxyears['year_x']))
# maxyear = dict(zip(minmaxyears['business_id'],
#                    minmaxyears['year_y']))
# #add that to the business dataset
# arizonafoodbusiness['minyear'] = arizonafoodbusiness['business_id'].
 ↪map(minyear)
# arizonafoodbusiness['maxyear'] = arizonafoodbusiness['business_id'].
 ↪map(maxyear)

# users profile from Arizona
arizonauserprofile = user['user_id'].isin(userarizona)
# elite users on Yelp
eliteusers = user[user['elite'] != 'None']
```

```
#getting the start year for yelping since
user['startyear'] = user['yelping_since'].apply(lambda x: x[0:4])

#remove outlier - gZGsReGOVeX4uKViHTB9EQ in Arizona
arizonafoodbusiness = arizonafoodbusiness[arizonafoodbusiness['business_id'] !=␣
 ↪'gZGsReGOVeX4uKViHTB9EQ']

# checkin values of the arizona food business
arizonafoodbusiness['sumcheckin'] = arizonafoodbusiness['business_id'].
 ↪map(sumfoodcheckin)

closedazbusiness = arizonafoodbusiness[arizonafoodbusiness['is_open'] == 0]
openazbusiness = arizonafoodbusiness[arizonafoodbusiness['is_open'] == 1]


#labeling review positive, negative, or neutral
def positiveornegative(star):
    if star > 3:
        return 'positive'
    elif star < 3:
        return 'negative'
    else:
        return 'neutral'
arizonafoodreviews['label'] = arizonafoodreviews['stars'].map(positiveornegative)

import gc
gc.collect()

#the code goes 70x faster when it's cached right here than to be called␣
 ↪repeatedly
stopw = stopwords.words('english')
# remove punctuations, lowercase,
#remove numbers r'\d+
#remove stop words, and split string into a list of words
arizonafoodreviews['clean text'] = arizonafoodreviews['text'].apply(
    lambda word: re.sub(r'\d+', '', re.sub(r'[^\w\s]', '', word.replace(
    '\n', ' '))))).apply(
    lambda word: [w for w in word.lower().split(' ') if w not in stopw and w !=␣
 ↪''])

positiveazfoodreviews = arizonafoodreviews[arizonafoodreviews['label'] ==␣
 ↪'positive']
negativeazfoodreviews = arizonafoodreviews[arizonafoodreviews['label'] ==␣
 ↪'negative']

#countvectorizer accepts array of one string not array of array of strings
```

```
arizonafoodreviews['clean text string'] = arizonafoodreviews['clean text'].
 ↪apply(lambda x: ' '.join(x))
```

[ ]:

2 Feature Engineering

[11]:
```
%%time
#collect one string of reviews for each business

def onereviewset(business_id):
    df = arizonafoodreviews[arizonafoodreviews['business_id'] == business_id]
    return df['clean text string'].str.cat(sep=' ')
arizonafoodbusiness['text'] = arizonafoodbusiness['business_id'].
 ↪map(onereviewset)
```

```
CPU times: user 16min 18s, sys: 36.1 s, total: 16min 55s
Wall time: 18min 4s
```

[12]:
```
%%time
from sklearn.feature_extraction.text import TfidfVectorizer

TV = TfidfVectorizer()

#Construct the required TF-IDF matrix by fitting and transforming the data
TV_matrix = TV.fit_transform(arizonafoodbusiness['text'])

#Output the shape of tfidf_matrix
TV_matrix.shape

# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel

# Compute the cosine similarity matrix
cosine_sim = linear_kernel(TV_matrix, TV_matrix)
```

```
CPU times: user 3min 4s, sys: 7.14 s, total: 3min 11s
Wall time: 3min 19s
```

[13]:
```
idtoindex = {}
num = 0
for i in arizonafoodbusiness['business_id']:
    idtoindex[str(i)] = num
    num += 1
indextoid = list(enumerate(idtoindex.keys()))
```

3 Recommendation System using Cosine Simiilarity and Content Based

Cosine similarity - takes what is most similar among the restaurants. Thus, we need to get features for each restaurant

Term Frequency-Inverse Document Frequency - Reduce the importance of the words that occur very frequently in the document

3.1 Most Basic Recommendation System

[14]:
```
def cosinerecommender(business_id):
    '''return the business_id, names of the
    top 10 most similar restaurants to this business_id.

    does not exclude same name of the business such as McDonalds'''
    #get position of the cosine_sim
    index = idtoindex[business_id]
    #enumerate counts the position of the list and acts as an iterator
    #get positions of all the values in the cosine_sim matrix for that␣
 ↪business_id
    values = list(enumerate(cosine_sim[index]))
    #sort values for top 10 recommendations
    # use key = lambda x: x[0] to sort the values not the indices
    # use reverse = True to get descending order
    sortedvalues = sorted(values, key = lambda x: x[1], reverse = True)
    #exclude the first one because that one is the identical as the business_id
    otherindices = [i[0] for i in sortedvalues[1:11]]
    businessesid = [indextoid[num][1] for num in otherindices]
    #return names of the restaurants
    return [(k,arizonafoodbusiness[arizonafoodbusiness['business_id'] ==␣
 ↪k]['name'].iloc[0]) for k in businessesid]
cosinerecommender('0kD9yJIB2cmvOh8-WbcO_w')
arizonafoodbusiness['Recommender1'] = arizonafoodbusiness['business_id'].
 ↪map(cosinerecommender)
```

Analysis: This shows based on the cosine similarity method of the text that people have the same opinion of the business as other businesses in the chain. For example, the text analysis shows that McDonald's Yelp Reviews are pretty similar.

In addition, When choosing a Greek restaurant, it would pop up other greek restaurants. Or when given a poke restaurant such as 'Poki Bar Central', it returns results such as 'Ocean Poke Co', 'Ahipoki Bowl'. This is a really good breakthrough or good recommender system if you're trying to find food with the same cuisine. And this is all based on text analysis. I didn't classify whether it was a poke restaurant in the count vectorizer.

I should extract the most common words in text by cuisine to see why these recommendations are appearing as such.

I should also create a recommender that recommends other food besides the same business name. For example, if you like McDonald's, perhaps you will like Wendy's as well.

[15]:
```
%%time
#clean the business name into simple form

arizonafoodbusiness['lowername'] = arizonafoodbusiness['name'].apply(lambda x: x.
 ↪replace("'", "").lower())
```

```
CPU times: user 14 ms, sys: 6.4 ms, total: 20.4 ms
Wall time: 27.6 ms
```

3.2 Revision 1: Recommend different businesses

[16]:
```
# Removing the same name business and recommend different businesses
def recommenddiffbus(business_id):
    '''return the business_id, names of the
    top 10 most similar restaurants to this business_id.

    does not include same name of the business such as McDonalds'''

    #get name of business_id
    name = arizonafoodbusiness[arizonafoodbusiness['business_id'] ==␣
 ↪business_id]['lowername'].iloc[0]
    #list of business id with that name
    listofbusinessid = arizonafoodbusiness[arizonafoodbusiness['lowername'] ==␣
 ↪name]['business_id']
    samenameindices = [idtoindex.get(k) for k in listofbusinessid]

#     #get position of the cosine_sim
    index = idtoindex[business_id]

    values = list(enumerate(cosine_sim[index]))
    sortedvalues = sorted(values, key = lambda x: x[1], reverse = True)

    #eliminate the same name
    differentindices = [i[0] for i in sortedvalues if i[0] not in␣
 ↪samenameindices]

#     #pick the first 10 excluding the first one
    otherindices = differentindices[1:11]
    businessesid = [indextoid[num][1] for num in otherindices]
#     #return names of the restaurants
    return [(k,arizonafoodbusiness[arizonafoodbusiness['business_id'] ==␣
 ↪k]['name'].iloc[0]) for k in businessesid]
# recommenddiffbus('0kD9yJIB2cmv0h8-WbcO_w')
recommenddiffbus('rDMptJYWtnMhpQu_rRXHng')
```

[16]:
```
[('B0785SPM9b805PeWezavmQ', '"Taco Bell"'),
 ('KGiObVBRve0oEyMBfOywcQ', '"Church\'s Chicken"'),
```

```
('hcpDtcyZ3bcKBgQeHkayvQ', '"Sonic Drive-In"'),
('cUmYDqGhBLjH2IbZ0GK07Q', '"Jack in The Box"'),
('njiPpJhy94Ojq91Bev6ckA', '"Panda Express"'),
('GdV24nwlR29MPqNXFglZnQ', '"Wendy\'s"'),
('6c4AW8WTpTA_xZYOrJx3Eg', '"KFC"'),
('DdlJsYusOm6iudcm77CPxA', '"Streets of New York"'),
('hLrrgQwn-K-IPI8g2etAoQ', '"Morning Squeeze"'),
('5PwYiNSkeiwmrnyOBj5IMQ', '"Carl\'s Jr"')]
```

3.3 Revision 2: Recommend different businesses and different cuisine

[18]:
```
# Removing the same name business and recommend different businesses
def recommenddiffbusiness2(business_id):
    '''return the business_id, names of the
    top 10 most similar restaurants to this business_id.

    does not include same name of the business such as McDonalds
    does not include same cuisine'''

    #get name of business_id
    name = arizonafoodbusiness[arizonafoodbusiness['business_id'] ==␣
 ↪business_id]['lowername'].iloc[0]
    # get main cuisine of business_id
    cuisine = arizonafoodbusiness[arizonafoodbusiness['business_id'] ==␣
 ↪business_id]['maincuisine'].iloc[0]
    #list of business id with that name
    listofbusinessid = arizonafoodbusiness[arizonafoodbusiness['lowername'] ==␣
 ↪name]['business_id']
    samenameindices = [idtoindex.get(k) for k in listofbusinessid]

    #list of businesses with same cuisine
    listofbusinessid2 = arizonafoodbusiness[arizonafoodbusiness['maincuisine']␣
 ↪== cuisine]['business_id']
    samenameindices2 = [idtoindex.get(k) for k in listofbusinessid2]


#     #get position of the cosine_sim
    index = idtoindex[business_id]

    values = list(enumerate(cosine_sim[index]))
    sortedvalues = sorted(values, key = lambda x: x[1], reverse = True)


    #eliminate the same name
    differentindices = [i[0] for i in sortedvalues if i[0] not in␣
 ↪samenameindices and i[0] not in samenameindices2]
```

```
#     pick the first 10 excluding the first one
    otherindices = differentindices[1:11]
    businessesid = [indextoid[num][1] for num in otherindices]
#     #return names of the restaurants
    return [(k,arizonafoodbusiness[arizonafoodbusiness['business_id'] ==␣
 ↪k]['name'].iloc[0]) for k in businessesid]
# recommenddiffbusiness('0kD9yJIB2cmv0h8-WbcO_w')
recommenddiffbusiness2('rDMptJYWtnMhpQu_rRXHng')
```

[18]:
```
[('B0785SPM9b805PeWezavmQ', '"Taco Bell"'),
 ('cUmYDqGhBLjH2IbZOGK07Q', '"Jack in The Box"'),
 ('njiPpJhy94Ojq91Bev6ckA', '"Panda Express"'),
 ('DdlJsYusOm6iudcm77CPxA', '"Streets of New York"'),
 ('hLrrgQwn-K-IPI8g2etAoQ', '"Morning Squeeze"'),
 ('5PwYiNSkeiwmrnyOBj5IMQ', '"Carl\'s Jr"'),
 ('LpdW1vh1AVm5QznEL5j03g', '"Panda Express"'),
 ('XGLGvB8S6-Anmt7gZkPLkQ', '"Dunkin\' Donuts"'),
 ('bVXGCxKYvylArU9JkPkkRQ', '"Starbucks"'),
 ('Q4pg6LcRCYMlml-5jg4Z5Q', '"Breakfast Kitchen Bar"')]
```

3.4 Revision 3: Recommend different businesses but in same cuisine

[19]:
```
# Removing the same name business and recommend different businesses
def recommenddiffbusiness3(business_id):
    '''return the business_id, names of the
    top 10 most similar restaurants to this business_id.

    does not include same name of the business such as McDonalds
    does include same cuisine'''

    #get name of business_id
    name = arizonafoodbusiness[arizonafoodbusiness['business_id'] ==␣
 ↪business_id]['lowername'].iloc[0]
    # get main cuisine of business_id
    cuisine = arizonafoodbusiness[arizonafoodbusiness['business_id'] ==␣
 ↪business_id]['maincuisine'].iloc[0]
    #list of business id with that name
    listofbusinessid = arizonafoodbusiness[arizonafoodbusiness['lowername'] ==␣
 ↪name]['business_id']
    samenameindices = [idtoindex.get(k) for k in listofbusinessid]

    #list of businesses with same cuisine
    listofbusinessid2 = arizonafoodbusiness[arizonafoodbusiness['maincuisine']␣
 ↪== cuisine]['business_id']
    samenameindices2 = [idtoindex.get(k) for k in listofbusinessid2]
```

```
#     #get position of the cosine_sim
    index = idtoindex[business_id]

    values = list(enumerate(cosine_sim[index]))
    sortedvalues = sorted(values, key = lambda x: x[1], reverse = True)


    #eliminate the same name
    differentindices = [i[0] for i in sortedvalues if i[0] not in␣
 ↪samenameindices and i[0] in samenameindices2]

#     pick the first 10 excluding the first one
    otherindices = differentindices[1:11]
    businessesid = [indextoid[num][1] for num in otherindices]

#     for k in businessesid:
#         print(arizonafoodbusiness[arizonafoodbusiness['business_id'] ==␣
 ↪k]['maincuisine'].iloc[0])

#     #return names of the restaurants
    return [(k,arizonafoodbusiness[arizonafoodbusiness['business_id'] ==␣
 ↪k]['name'].iloc[0]) for k in businessesid]
# recommenddiffbusiness('0kD9yJIB2cmv0h8-WbcO_w')
recommenddiffbusiness3('rDMptJYWtnMhpQu_rRXHng')
```

```
Fast Food
Fast Food
Fast Food
Fast Food
Fast Food
Fast Food
Fast Food
Fast Food
Fast Food
Fast Food
```

[19]:
```
[('hcpDtcyZ3bcKBgQeHkayvQ', '"Sonic Drive-In"'),
 ('GdV24nwlR29MPqNXFglZnQ', '"Wendy\'s"'),
 ('6c4AW8WTpTA_xZYOrJx3Eg', '"KFC"'),
 ('b5kC2XepQ1-6qCDpvtoxxA', '"Taco Bell"'),
 ('3egp_Vx6nZCLyJVdCCFywQ', '"Popeyes Louisiana Kitchen"'),
 ('anhk98859Rmo3mL6X41jbg', '"Panda Express"'),
 ('GZ_523PIvW3ifTwn0SAJJw', '"Sonic Drive-In"'),
 ('uYDScUhTHjtnnxmzy4fgPQ', '"Jack in the Box"'),
 ('JQiYvTuxwNxebyr5QIQy6Q', '"Jack in the Box"'),
 ('VTq6g0XGokIS9TPLwHO-hg', '"KFC / A&W"')]
```

4 Recommendation System using Collaborative Filtering and Users

[5]:
```
%%time
#business id to name
businessidtoname = dict(zip(arizonafoodbusiness['business_id'],␣
 ↪arizonafoodbusiness['lowername']))

# add business name to AZ food reviews
arizonafoodreviews['business_name'] = arizonafoodreviews['business_id'].
 ↪map(businessidtoname)


# def historyofreviews(user):
#     '''takes in a user_id and returns dictionary of business id to array of␣
 ↪stars for the user'''
#     userreviews = arizonafoodreviews[arizonafoodreviews['user_id'] == user]
#     return dict(zip(userreviews['business_id'], userreviews['stars'].values))


def historyofusers(business_name):
    '''takes in a business id
    returns dictionary of user_id to array of stars for that business_id'''
    reviews = arizonafoodreviews[arizonafoodreviews['business_name'] ==␣
 ↪business_name]
    return dict(zip(reviews['user_id'], reviews['stars'].values))

arizonafoodbusiness['users and their ratings'] =␣
 ↪arizonafoodbusiness['lowername'].map(historyofusers)

azuser = user[arizonauserprofile]
a = azuser[azuser['review_count']>1]

def matrix(x):
    w = np.zeros(a.shape[0])
    num = 0
#     a = azuser[azuser['review_count']>1]
    h = x
    for user in a['user_id']:
        if user in h.keys():
            np.put(w, num, h.get(user))
        num +=1
    return w

arizonafoodbus = arizonafoodbusiness[arizonafoodbusiness['review_count'] > 100 &␣
 ↪arizonafoodbusiness['lowername'].isin(arizonafoodbusiness['lowername'].
 ↪unique())]
```

```
arizonafoodbus['matrix'] = arizonafoodbus['users and their ratings'].map(matrix)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  """

CPU times: user 30min 30s, sys: 32.8 s, total: 31min 3s
Wall time: 32min 18s
```

[ ]:
```
# %%time
# gc.collect()
# rm = np.matrix(arizonafoodbus['matrix'].to_list())
```

[72]:
```
sizebusreviews = arizonafoodreviews.groupby('business_id').count().reset_index()
dataforcf = arizonafoodreviews[arizonafoodreviews['business_id'].
 ↪isin(sizebusreviews[sizebusreviews['review_id'] > 600]['business_id'])]
```

[ ]:
```
# pd.pivot_table(dataforcf[['user_id', 'business_id', 'stars']], index =␣
 ↪'user_id', columns = 'business_id', values = 'stars')
```

[ ]:
```
17839
```