Yelp Sentiment Analysis

September 4, 2019

1 SENTIMENT OF REVIEWS

Word PreProcessing/Cleaning up text

1. Remove punctuations and special characters
2. Remove /n and remove numbers
3. Remove stop words and one-two letter words
4. Stem the words
5. Split the string into words in an array

Extracting Positive/Negative Words 1. Apply the positive and negative sets to the text
Split into Negative and Positive Reviews
Reviews with more than 3 stars are considered positive.
Reviews with less than 3 stars are considered negative.
Review with 3 stars are neutral.
TABLE OF CONTENTS

[1]:
```
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
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
from shapely.geometry import Point
# from arcgis.gis import GIS
```

[2]:
```
%%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 19s, sys: 1min 7s, total: 2min 27s
Wall time: 6min 45s
```

[ ]:
```
%%time
# 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)

# #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)


# 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 - gZGsReG0VeX4uKViHTB9EQ in Arizona
arizonafoodbusiness = arizonafoodbusiness[arizonafoodbusiness['business_id'] !=␣
 ↪'gZGsReG0VeX4uKViHTB9EQ']

# 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']
```

2 Positive and Negative Preprocessing

[5]:
```
# positive and negative text files and turn into set
positive = pd.read_csv('positive-words.txt', header = None)
positive = set(positive[0])
positivelisttoadd = ['razzledazzle', 'to die for', 'buttery']
for word in positivelisttoadd:
    positive.add(word)
negative = pd.read_csv('negative-words.txt', header = None)
negative = set(negative[0])
negativelisttoremove = ['wicked', 'die']
for w in negativelisttoremove:
    negative.remove(w)
negativelisttoadd = ['never', 'shitty']
for w in negativelisttoremove:
    negative.add(w)
```

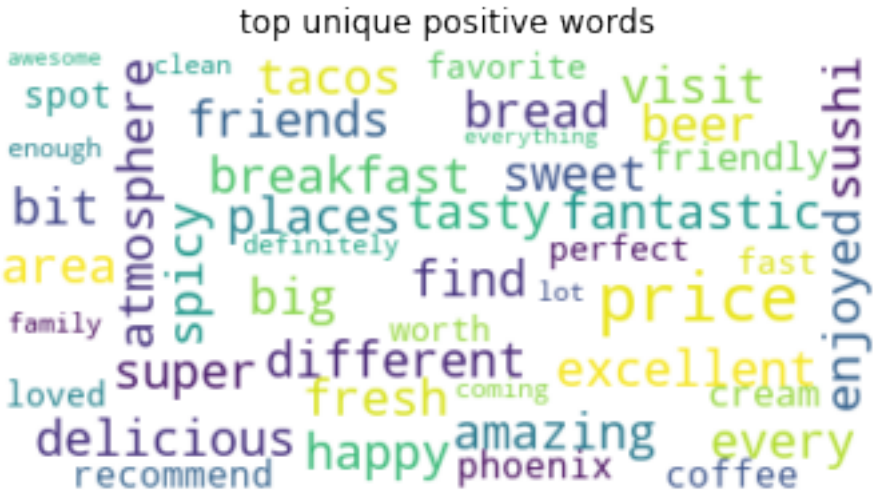
```
positiveazfoodreviews = arizonafoodreviews[arizonafoodreviews['label'] ==␣
 ↪'positive']
negativeazfoodreviews = arizonafoodreviews[arizonafoodreviews['label'] ==␣
 ↪'negative']

# positive words
arizonafoodreviews['positivewords'] = arizonafoodreviews['clean text'].apply(
    lambda w: [h for h in w if h in positive])
#negative words
arizonafoodreviews['negativewords'] = arizonafoodreviews['clean text'].apply(
    lambda w: [h for h in w if h in negative])
```

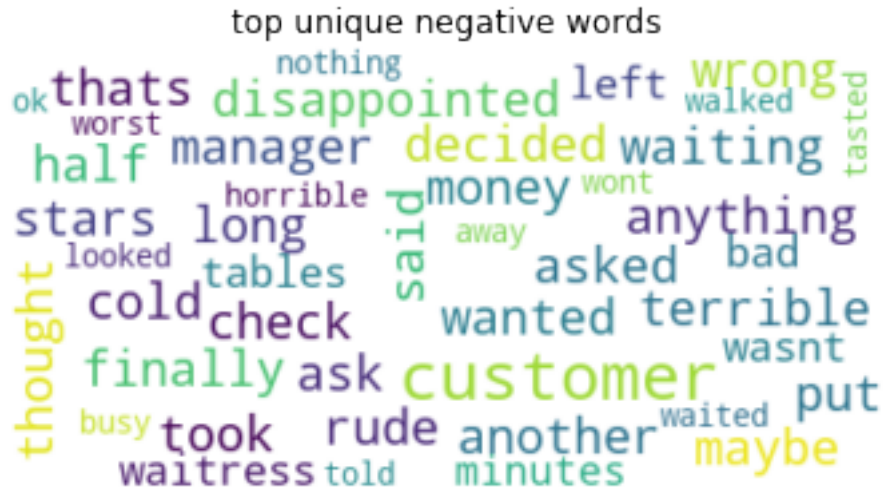
```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:21:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:24:
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
```

3 Positive and Negative Word Clouds

[6]:
```
negwordcount = Counter(' '.join(negativeazfoodreviews['clean text'].apply(
lambda z: ' '.join(z))).split(' '))
top150negwords = [w[0] for w in negwordcount.most_common(150)]

poswordcount = Counter(' '.join(positiveazfoodreviews['clean text'].apply(
lambda z: ' '.join(z))).split(' '))
top150poswords = [w[0] for w in poswordcount.most_common(150)]

uniqueposwords = list(set(top150poswords) - set(top150negwords))
uniquenegwords = list(set(top150negwords) - set(top150poswords))

wordcloud = WordCloud(max_font_size=30, max_words=150, background_color="white").
 ↪generate(' '.join(uniqueposwords))
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title('top unique positive words')
```

```
plt.show()


wordcloud = WordCloud(max_font_size=30, max_words=150, background_color="white").
 ↪generate(' '.join(uniquenegwords))
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title('top unique negative words')
plt.show()
```





4 Positive and Negative Bigrams and Trigrams

[7]:
```
negstring = []
for text in negativeazfoodreviews['clean text']:
    for word in text:
        negstring.append(word)
```

[8]:
```
%%time
import nltk
bigrams = nltk.collocations.BigramAssocMeasures()
trigrams = nltk.collocations.TrigramAssocMeasures()

negbigramFinder = nltk.collocations.BigramCollocationFinder.from_words(negstring)
negtrigramFinder = nltk.collocations.TrigramCollocationFinder.
 ↪from_words(negstring)

negbigram_freq = negbigramFinder.ngram_fd.items()
negbigramFreqTable = pd.DataFrame(list(negbigram_freq),␣
 ↪columns=['bigram','freq']).sort_values(by='freq', ascending=False)

negtrigram_freq = negtrigramFinder.ngram_fd.items()
negtrigramFreqTable = pd.DataFrame(list(negtrigram_freq),␣
 ↪columns=['trigram','freq']).sort_values(by='freq', ascending=
False)
```

```
CPU times: user 2min 18s, sys: 8.08 s, total: 2min 26s
Wall time: 2min 44s
```

Negative Bigram and Trigram Collocation

Bigram:

```
Customer Service is the most mentioned bigram phrase in the entire arizona food reviews.

Since it's a negative review marked by lower than 3 stars, the bigram do not include the negativ

Thus, for negative reviews, bigrams are not that helpful in understanding sentiment.
```

Trigram:

```
Service: took minutes (to) get, worst service eer,  poor customer service, horrible customer ser

Food:

Experience: never go back,  wont going back,  won't go back,  never come back,don't waste time,

Price: waste time and money

Trigrams are helpful in understanding negative sentiment.
```

[9]:
```
posstring = []
for text in positiveazfoodreviews['clean text']:
    for word in text:
        posstring.append(word)
```

[10]:
```
%%time

bigrams = nltk.collocations.BigramAssocMeasures()
trigrams = nltk.collocations.TrigramAssocMeasures()

posbigramFinder = nltk.collocations.BigramCollocationFinder.from_words(posstring)
postrigramFinder = nltk.collocations.TrigramCollocationFinder.
 ↪from_words(posstring)

posbigram_freq = posbigramFinder.ngram_fd.items()
posbigramFreqTable = pd.DataFrame(list(posbigram_freq),␣
 ↪columns=['bigram','freq']).sort_values(by='freq', ascending=False)

postrigram_freq = postrigramFinder.ngram_fd.items()
postrigramFreqTable = pd.DataFrame(list(postrigram_freq),␣
 ↪columns=['trigram','freq']).sort_values(by='freq', ascending=
False)
```

```
CPU times: user 5min 49s, sys: 51.7 s, total: 6min 40s
Wall time: 7min 48s
```

posbigram: happy hour, first time, great food, really good, love place, great service, great place,good food,can't wait, highly recommend, good service, friendly staff, mexican food, ice cream, customer service, super friendly, last night,

postrigram: definitely come back, definitely go back, definitely coming back, highly recommend place, would highly recommend, would definitely recommend, staff always friendly, staff super friendly, one favorite places, can't go wrong

5 Tag Classification of Yelp Reviews

[24]:
```
# Tag classification of Yelp Reviews
negativeazfoodreviews['Service'] = negativeazfoodreviews['clean text'].apply(
    lambda h: True if 'service' in h else False)
# Tag classification of Yelp Reviews
negativeazfoodreviews['Money'] = negativeazfoodreviews['clean text'].apply(
    lambda h: True if any(word in h for word in ['money', 'expensive', 'pricey',␣
 ↪'cost']) else False)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3:
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
  This is separate from the ipykernel package so we can avoid doing imports
until
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:6:
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
```

[22]:
```
negativeazfoodreviews['Service'].value_counts()
```

[22]:
```
False    135166
True      72790
Name: Service, dtype: int64
```

[25]:
```
negativeazfoodreviews['Money'].value_counts()
```

[25]:
```
False    183103
True      24853
Name: Money, dtype: int64
```

53.9% of the negative reviews mention service.
13.57% of the negative reviews mention something related to the price and money

6 Predicting Sentiment

[4]:
```
#countvectorizer accepts array of one string not array of array of strings
arizonafoodreviews['clean text string'] = arizonafoodreviews['clean text'].
 ↪apply(lambda x: ' '.join(x))
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
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
```

[5]:
```
%%time

# Creating the Bag of Words model
#Bag of words scheme is the simplest way of converting text to numbers.
```

```
from sklearn.feature_extraction.text import CountVectorizer

#max features = the number of the most frequently occuring words in the dataset
# To extract max 1500 feature.
# "max_features" is attribute to
# experiment with to get better results
cv = CountVectorizer(max_features = 1500)

#The frequency of the word in the document will replace the actual word in the␣
 ↪vocabulary. If a word in the vocabulary is not found in the corresponding␣
 ↪document, the document feature vector will have zero in that place
#In the bag of words approach, each word has the same weight.

# X contains corpus (dependent variable)
X = cv.fit_transform(np.array(arizonafoodreviews['clean text string'])).toarray()
Y = arizonafoodreviews['stars']

#split into train and test sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=0)

from sklearn.ensemble import RandomForestClassifier

text_classifier = RandomForestClassifier(n_estimators=200, random_state=0)
text_classifier.fit(X_train, y_train)
predictions = text_classifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
print(accuracy_score(y_test, predictions))
```

```
[[18088  1484   838  1171  2023]
 [ 6593  2725  2878  3326  2493]
 [ 2569  1400  5051  9258  4580]
 [ 1086   289  1758 19927 25276]
 [  972   101   362  9390 71032]]
              precision    recall  f1-score   support

           1       0.62      0.77      0.68     23604
           2       0.45      0.15      0.23     18015
           3       0.46      0.22      0.30     22858
           4       0.46      0.41      0.44     48336
           5       0.67      0.87      0.76     81857
```

```
    accuracy                           0.60    194670
   macro avg       0.53      0.48      0.48    194670
weighted avg       0.57      0.60      0.57    194670

0.6001078748651564
CPU times: user 2h 31min 55s, sys: 42min 28s, total: 3h 14min 24s
Wall time: 6h 54min 39s
```

[6]:
```
%%time
cv = CountVectorizer(max_features = 1500)

X = cv.fit_transform(np.array(arizonafoodreviews['clean text string'])).toarray()

Y = arizonafoodreviews['label']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=0)

from sklearn.ensemble import RandomForestClassifier

text_classifier = RandomForestClassifier(n_estimators=200, random_state=0)
text_classifier.fit(X_train, y_train)
predictions = text_classifier.predict(X_test)

print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
print(accuracy_score(y_test, predictions))
```

```
[[ 32253    604   8762]
 [  6207   1570  15081]
 [  3058    522 126613]]
              precision    recall  f1-score   support

    negative       0.78      0.77      0.78     41619
     neutral       0.58      0.07      0.12     22858
    positive       0.84      0.97      0.90    130193

    accuracy                           0.82    194670
   macro avg       0.73      0.61      0.60    194670
weighted avg       0.80      0.82      0.78    194670

0.8241434222016746
CPU times: user 2h 15min 22s, sys: 4min 33s, total: 2h 19min 56s
Wall time: 2h 38min 32s
```

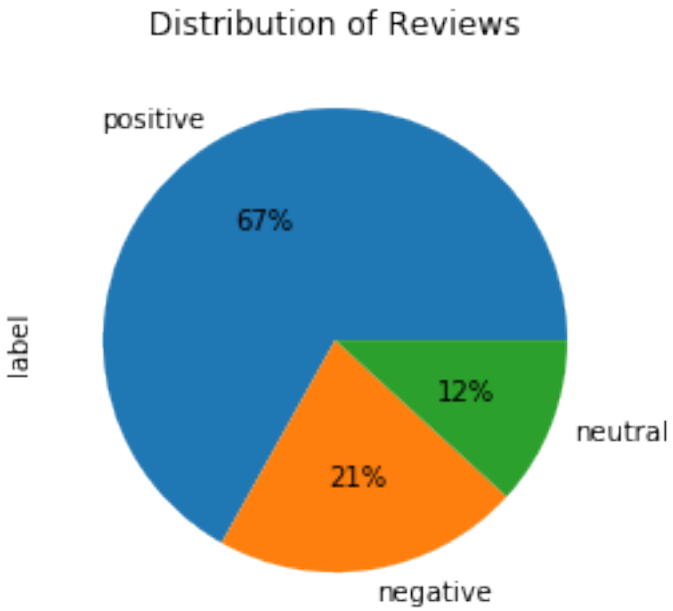
[7]:
```
%%time
cv = CountVectorizer(max_features = 1500)

X = cv.fit_transform(np.array(arizonafoodreviews[arizonafoodreviews['stars'] !=␣
 ↪3]['clean text string'])).toarray()

Y = arizonafoodreviews[arizonafoodreviews['stars'] != 3]['label']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=0)

from sklearn.ensemble import RandomForestClassifier

text_classifier = RandomForestClassifier(n_estimators=200, random_state=0)
text_classifier.fit(X_train, y_train)
predictions = text_classifier.predict(X_test)

print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
print(accuracy_score(y_test, predictions))
```

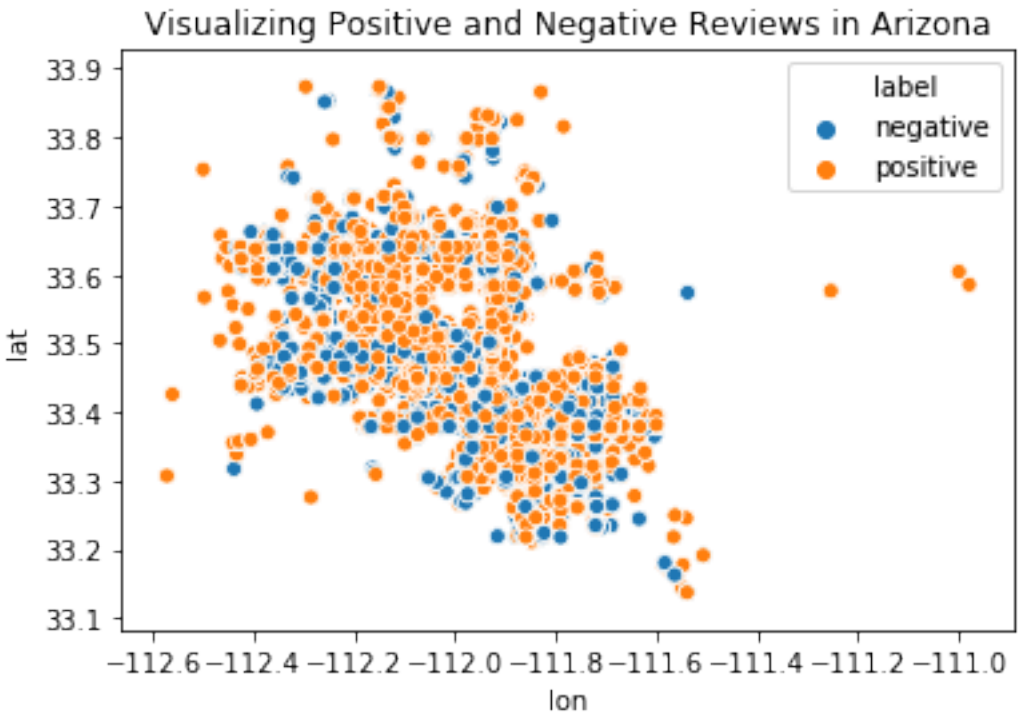
```
[[ 32850   8818]
 [  3399 126702]]
              precision    recall  f1-score   support

    negative       0.91      0.79      0.84     41668
    positive       0.93      0.97      0.95    130101

    accuracy                           0.93    171769
   macro avg       0.92      0.88      0.90    171769
weighted avg       0.93      0.93      0.93    171769

0.92887540825178
CPU times: user 2h 8min 39s, sys: 4min 57s, total: 2h 13min 37s
Wall time: 2h 21min 49s
```

7 Visualizing Sentiment

[10]:
```
arizonafoodreviews['label'].value_counts().plot(kind = 'pie',  autopct='%1.0f%%')
plt.title("Distribution of Reviews")
plt.show()

arizonafoodreviews = arizonafoodreviews[arizonafoodreviews['business_id'] !=␣
 ↪'gZGsReGOVeX4uKViHTB9EQ']
sns.scatterplot(x='lon', y = 'lat',
```

```
                hue = 'label', data =␣
 ↪arizonafoodreviews[arizonafoodreviews['label'] != 'neutral'])
plt.title('Visualizing Positive and Negative Reviews in Arizona');
```





[ ]:

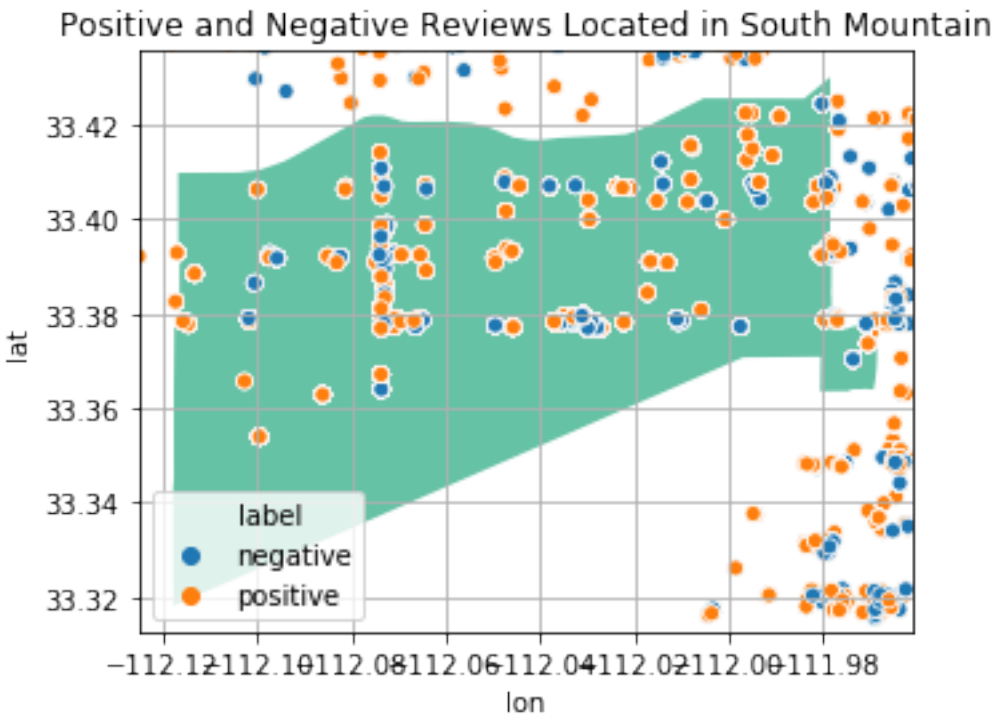
[5]:
```
%%time
arizona = geopandas.read_file('Villages/Villages.shp')
arizona.geometry = arizona.geometry.to_crs(epsg = 4326)
#coordinate reference system
crs = {'init': 'epsg:2868'}
#create a point geometry column using Shapely Point
arizonafoodreviews['geometry'] = arizonafoodreviews.apply(
    lambda h: Point(float(h['lon']), float(h['lat'])), axis = 1)
GEO_AZr= geopandas.GeoDataFrame(arizonafoodreviews, crs = crs, geometry =␣
 ↪arizonafoodreviews['geometry'])
```

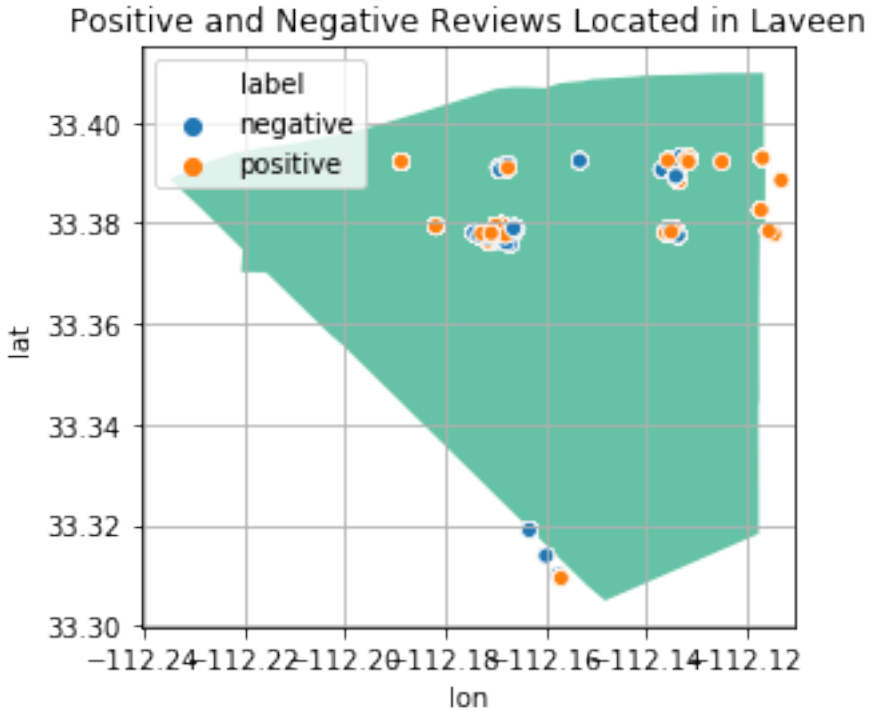
```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

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

CPU times: user 1min 17s, sys: 1min 18s, total: 2min 36s
Wall time: 4min 47s
```

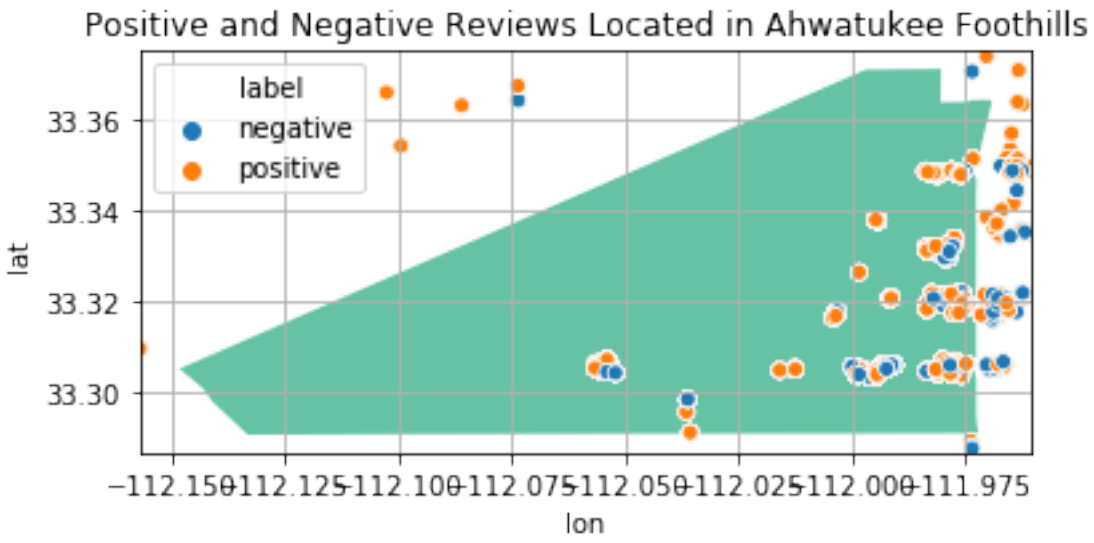
[28]:
```
for n in arizona['NAME']:
    neighborhood = arizona[arizona['NAME'] == n]
    ax = neighborhood.plot(column='NAME', cmap='Set2', legend = True)
    print(plt.xlim())
    print(plt.ylim())
    plt.xlim(plt.xlim())
    plt.ylim(plt.ylim())
    sns.scatterplot(x='lon',y= 'lat',hue='label', data =␣
 ↪GEO_AZr[GEO_AZr['label'] != 'neutral'])
    plt.title("Positive and Negative Reviews Located in " + n)
    plt.grid()
```

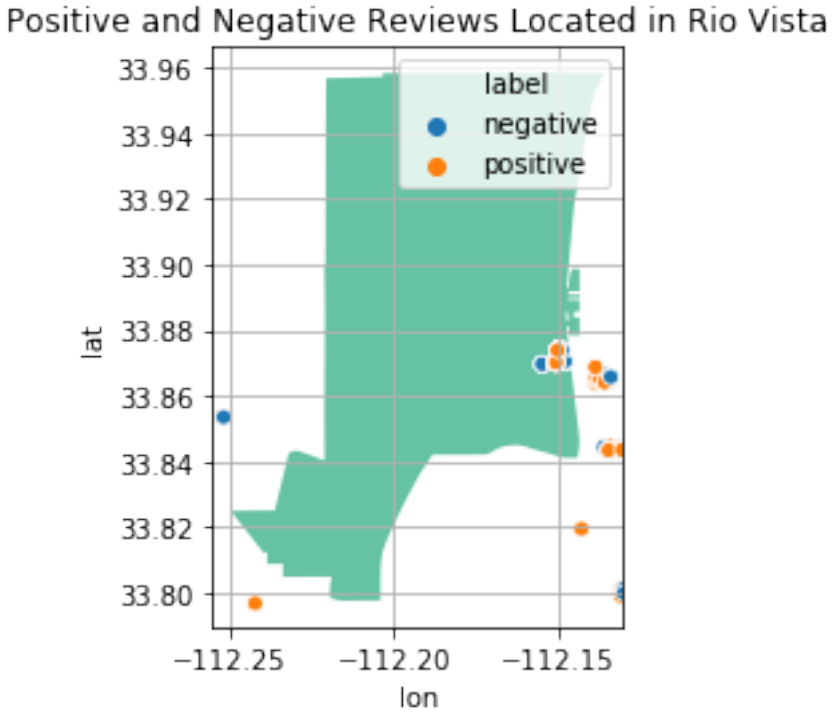
```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

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

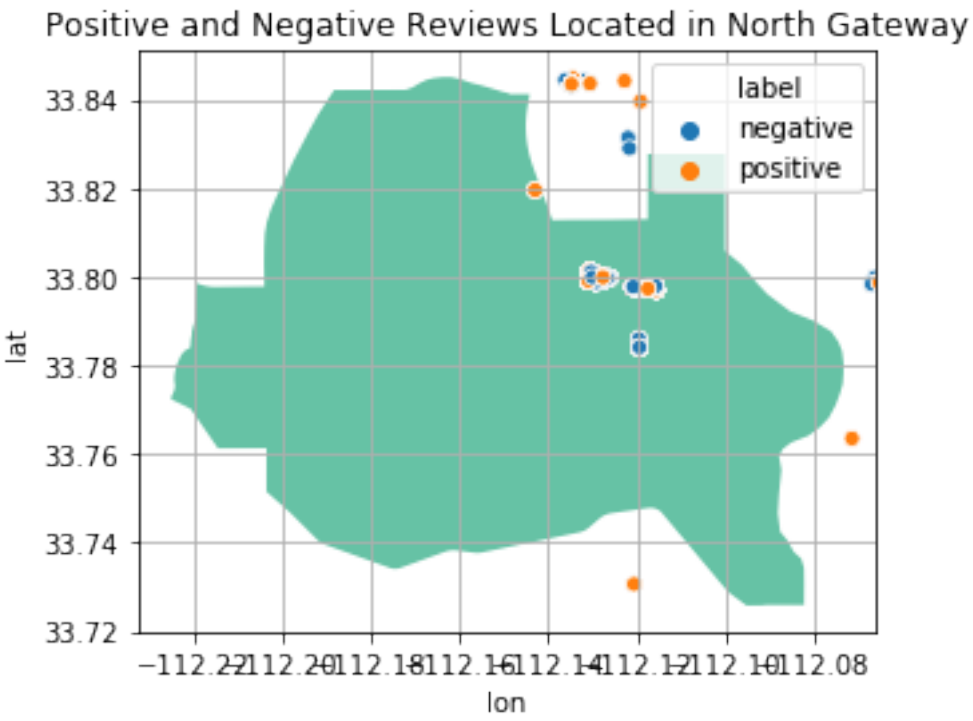
(-112.12523107535863, -111.96103736015073)
(33.31264575094429, 33.43568298013611)
(-112.2407514802972, -112.1104565899216)
(33.29968137088586, 33.415074795080024)
(-112.15719993178277, -111.9603019161533)
(33.28642384749294, 33.37492391967232)
(-112.2553002538437, -112.13017430849013)
(33.789663862725, 33.966235419565656)
(-112.2326437666651, -112.06582024984229)
(33.719908230576074, 33.85095927793099)
(-112.1088856905336, -111.9167738916414)
(33.629752487874185, 33.81475931221413)
(-112.21131717309875, -112.04037532176265)
(33.61126400459976, 33.75822945927427)
(-112.05988760943394, -111.91913273834292)
(33.54629165823559, 33.70082985623964)
(-112.17584089665412, -112.02094882494369)
(33.548452947569665, 33.6418988715334)
(-112.07151834013554, -111.93296710582977)
(33.429629354335404, 33.57349600384656)
(-112.15611258743654, -112.06065695049367)
(33.47322957258423, 33.55691626120545)
(-112.3348381508246, -112.09759701005333)
(33.45878839095518, 33.5254894804065)
(-112.11799251885353, -112.01119863441181)
(33.46380309247279, 33.50594061858192)
(-112.11644473655328, -111.97178934609452)
(33.40765100203165, 33.468685105315274)
(-112.29907893927793, -112.0899995523301)
(33.37776718386452, 33.4671122011608)

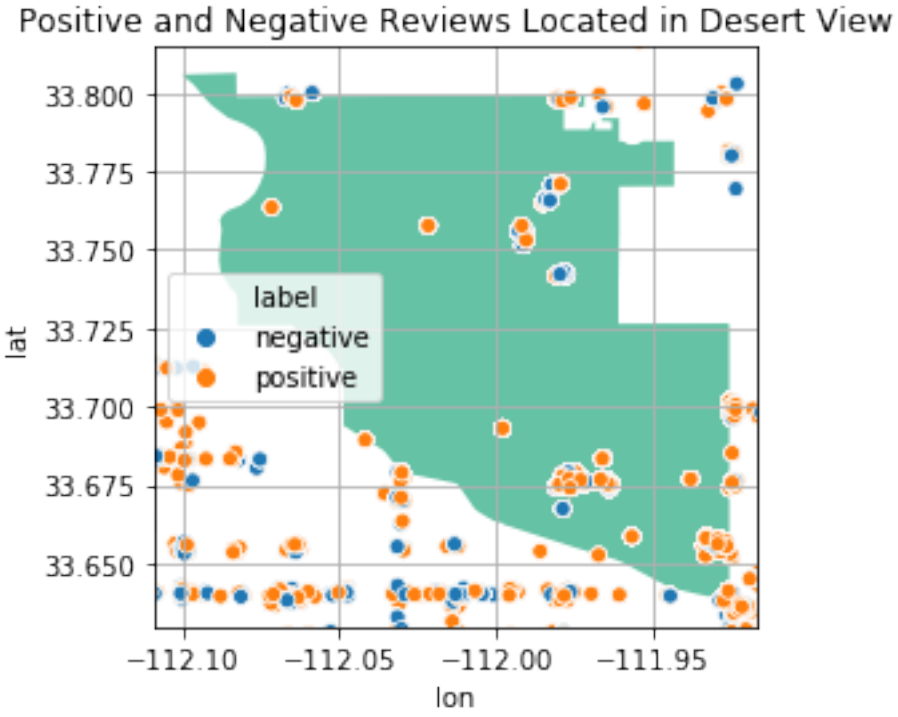

Positive and Negative Reviews Located in South Mountain
33.42
33.40
33.38
33.36
33.34
33.32
lat
label
negative
positive
−112.12
−112.10
−112.08
−112.06
−112.04
−112.02
−112.00
−111.98
lon

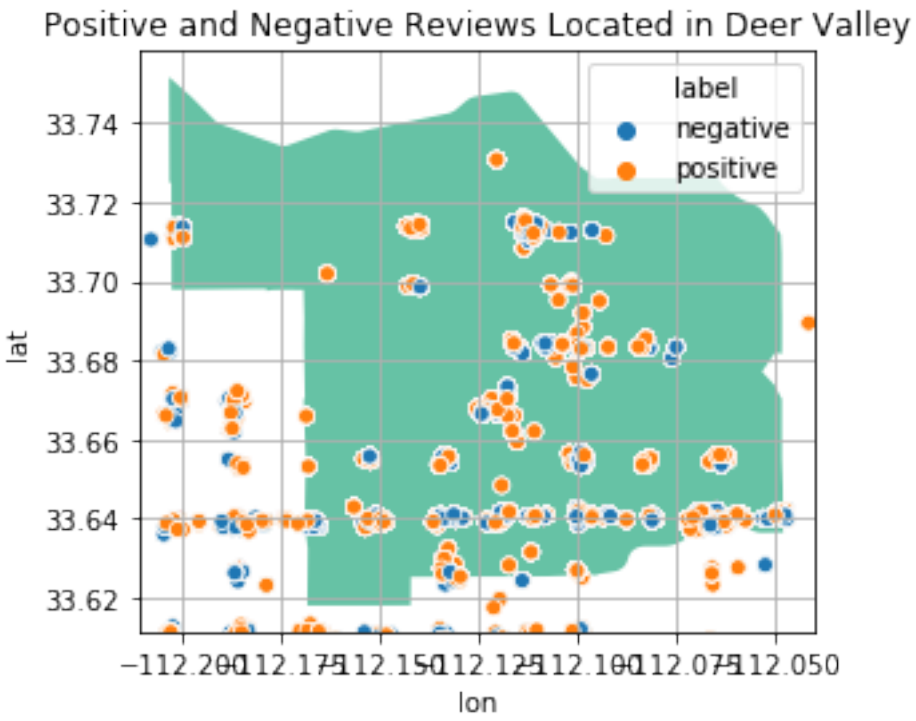

Positive and Negative Reviews Located in Laveen
33.40
33.38
33.36
33.34
33.32
33.30
lat
label
negative
positive
−112.24
−112.22
−112.20
−112.18
−112.16
−112.14
−112.12
lon

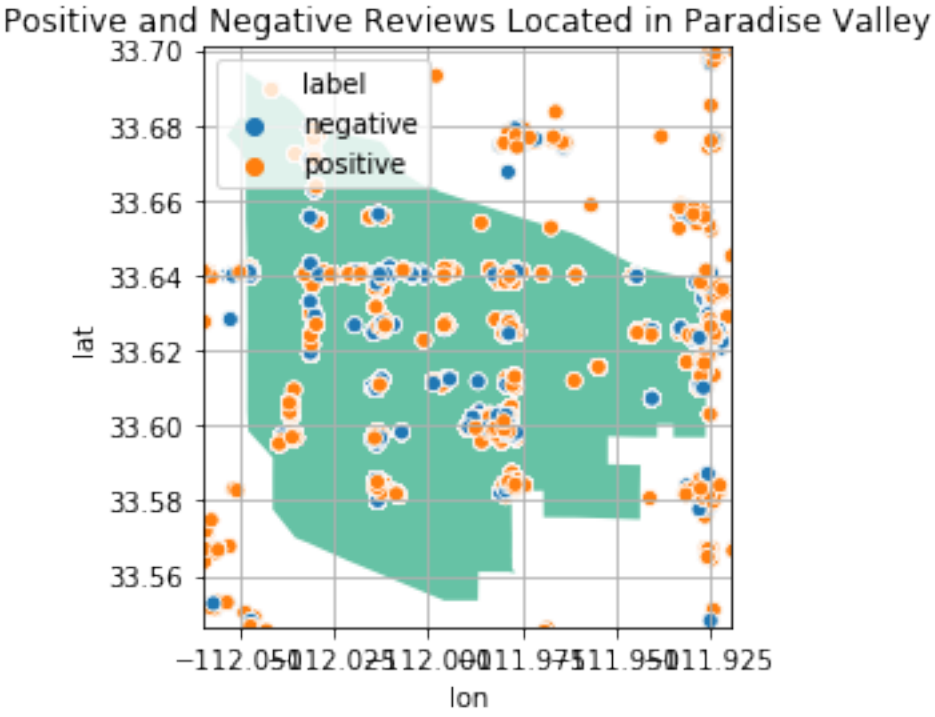

Positive and Negative Reviews Located in Ahwatukee Foothills
label
negative
positive
33.36
33.34
33.32
33.30
lat
−112.150
−112.125
−112.100
−112.075
−112.050
−112.025
−112.000
−111.975
lon

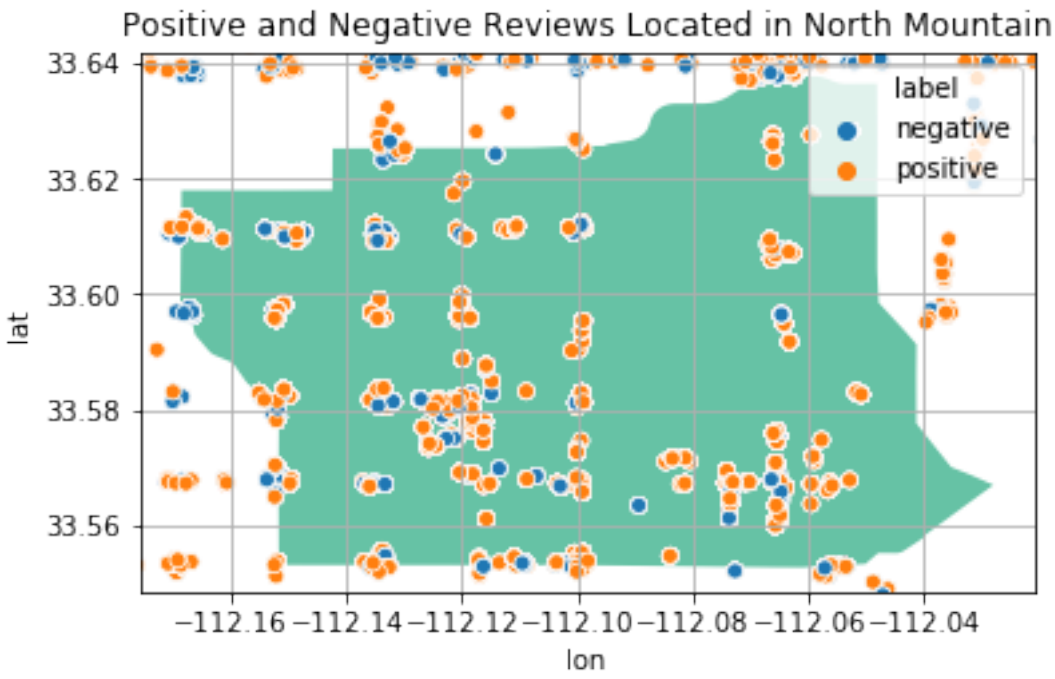

Positive and Negative Reviews Located in Rio Vista
label
negative
positive
33.96
33.94
33.92
33.90
33.88
33.86
33.84
33.82
33.80
lat
−112.25
−112.20
−112.15
lon

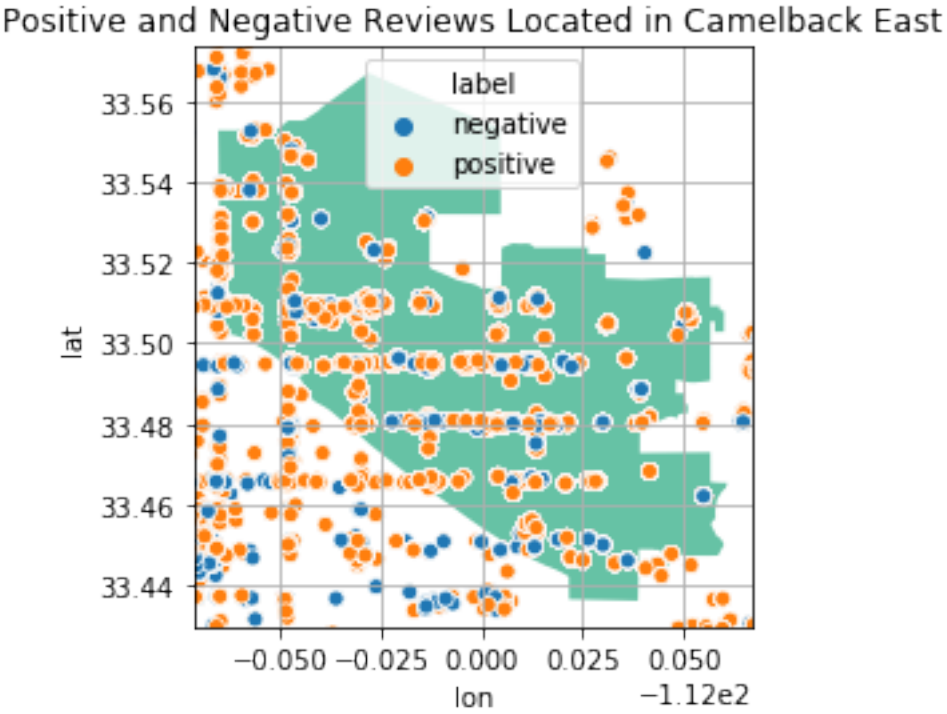

Positive and Negative Reviews Located in North Gateway
label
negative
positive
33.84
33.82
33.80
33.78
33.76
33.74
33.72
lat
lon

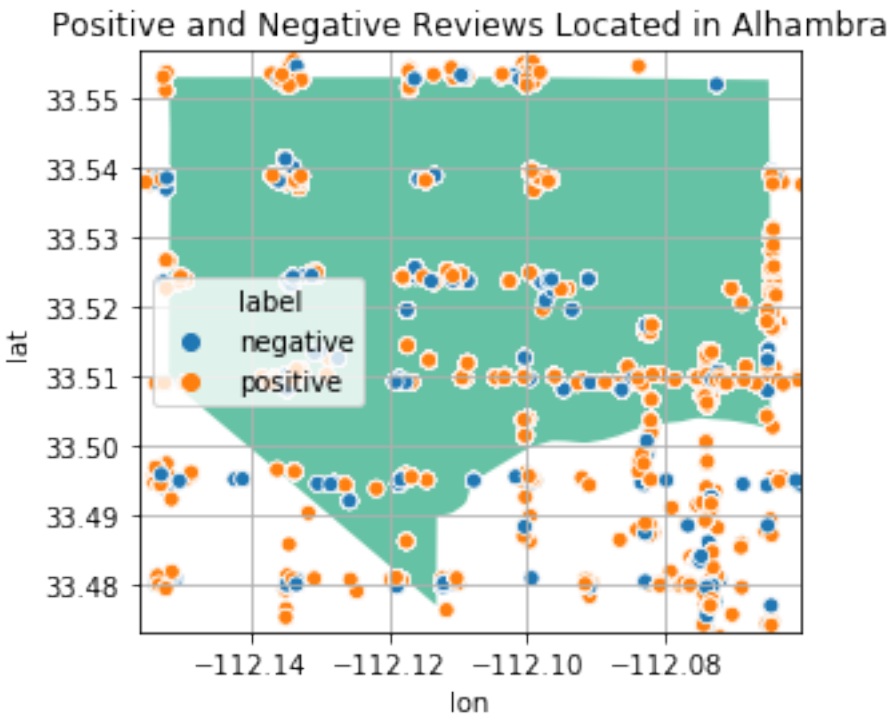

Positive and Negative Reviews Located in Desert View
label
negative
positive
33.800
33.775
33.750
33.725
33.700
33.675
33.650
lat
−112.10
−112.05
−112.00
−111.95
lon

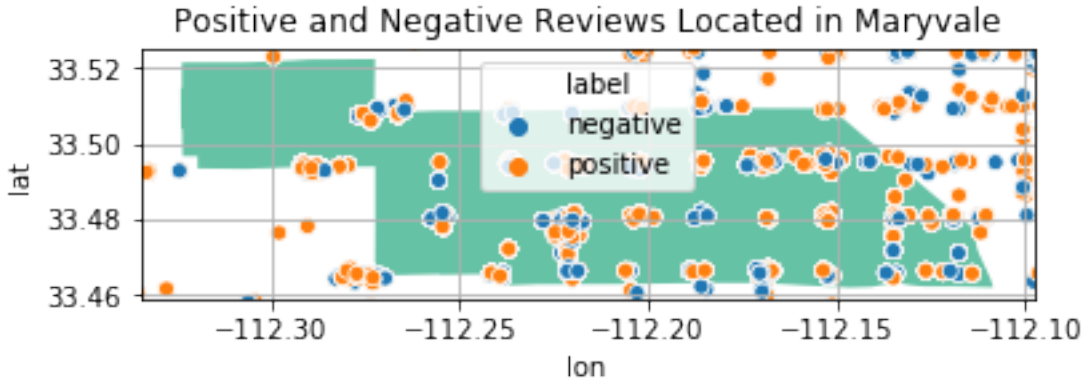

Positive and Negative Reviews Located in Deer Valley
label
negative
positive
33.74
33.72
33.70
33.68
33.66
33.64
33.62
lat
−112.200
112.175
112.150
112.125
112.100
112.075
112.050
lon

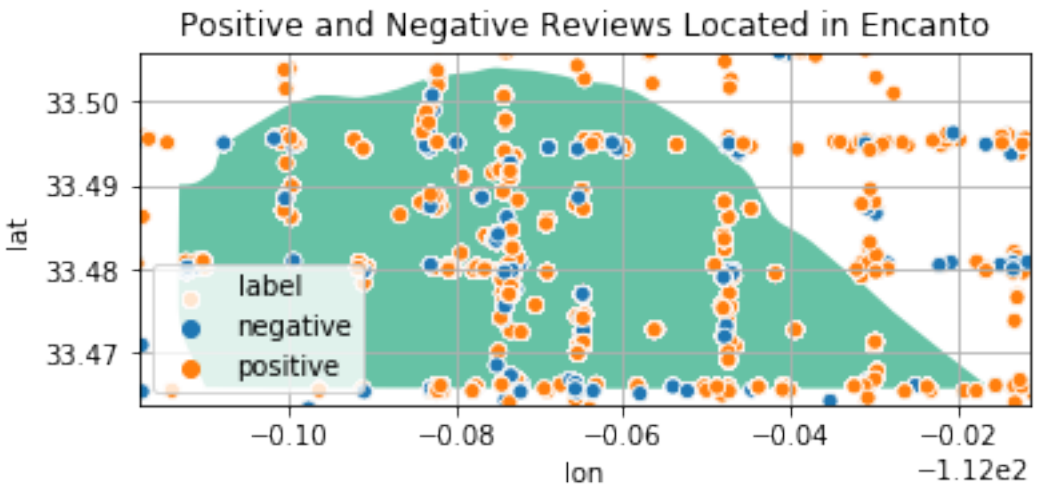

Positive and Negative Reviews Located in Paradise Valley
label
negative
positive
33.70
33.68
33.66
33.64
33.62
33.60
33.58
33.56
lat
−112.050
112.025
112.000
111.975
111.950
111.925
lon

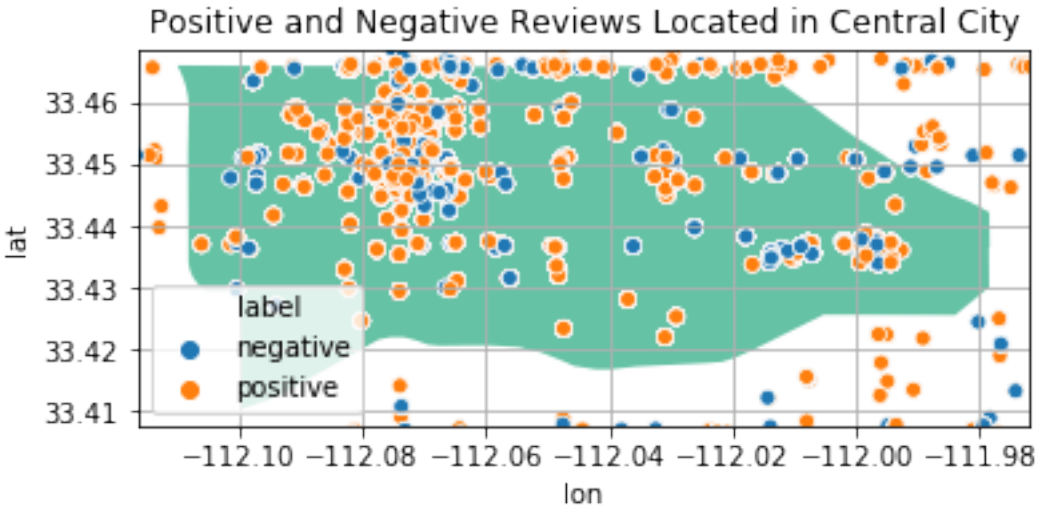

Positive and Negative Reviews Located in North Mountain
label
negative
positive
33.64
33.62
33.60
33.58
33.56
lat
−112.16
−112.14
−112.12
−112.10
−112.08
−112.06
−112.04
lon

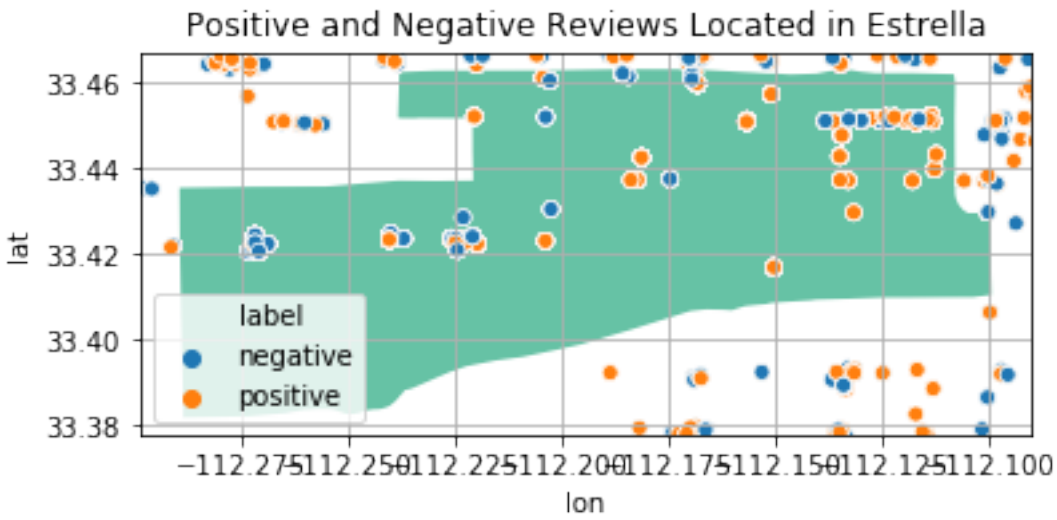

Positive and Negative Reviews Located in Camelback East
label
negative
positive
33.56
33.54
33.52
33.50
33.48
33.46
33.44
lat
−0.050
−0.025
0.000
0.025
0.050
−1.12e2
lon


Positive and Negative Reviews Located in Alhambra
33.55
33.54
33.53
33.52
33.51
33.50
33.49
33.48
lat
label
negative
positive
−112.14
−112.12
−112.10
−112.08
lon


Positive and Negative Reviews Located in Maryvale
33.52
33.50
33.48
33.46
lat
label
negative
positive
−112.30
−112.25
−112.20
−112.15
−112.10
lon


Positive and Negative Reviews Located in Encanto
33.50
33.49
33.48
33.47
lat
label
negative
positive
−0.10
−0.08
−0.06
−0.04
−0.02
−1.12e2
lon


Positive and Negative Reviews Located in Central City
33.46
33.45
33.44
33.43
33.42
33.41
lat
label
negative
positive
−112.10
−112.08
−112.06
−112.04
−112.02
−112.00
−111.98
lon

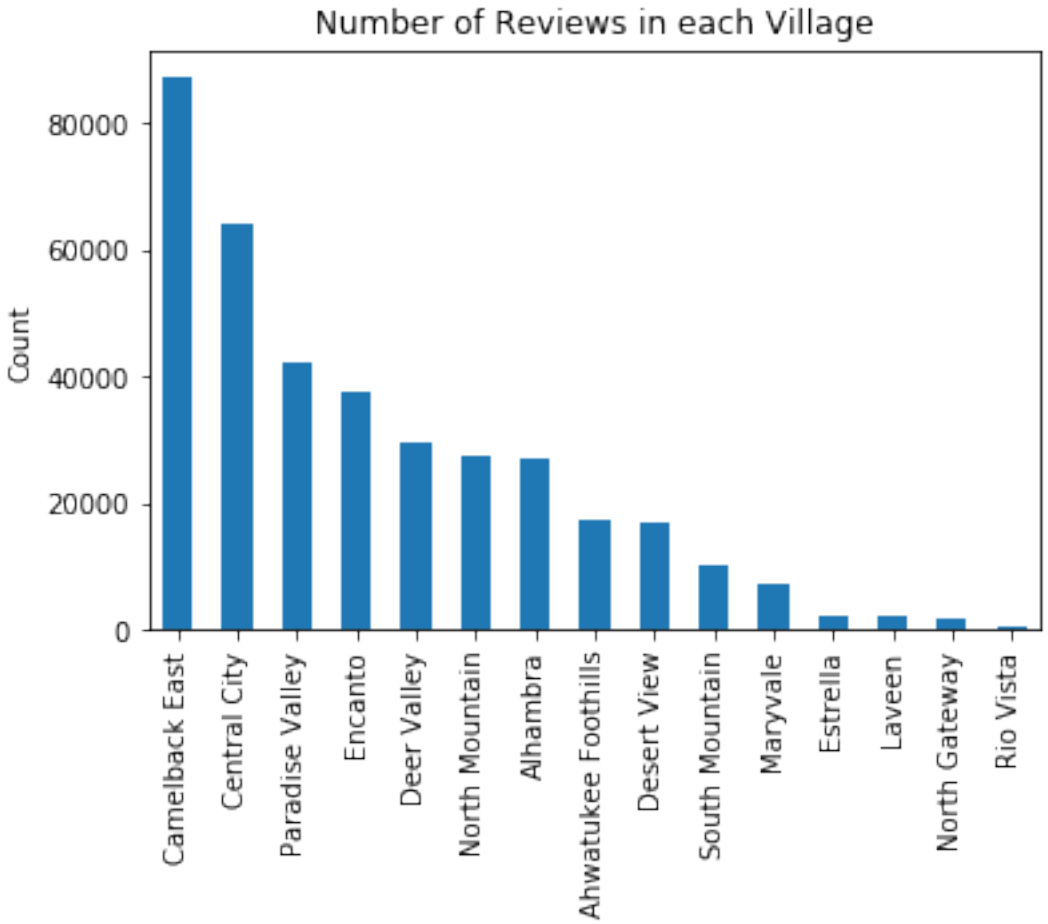


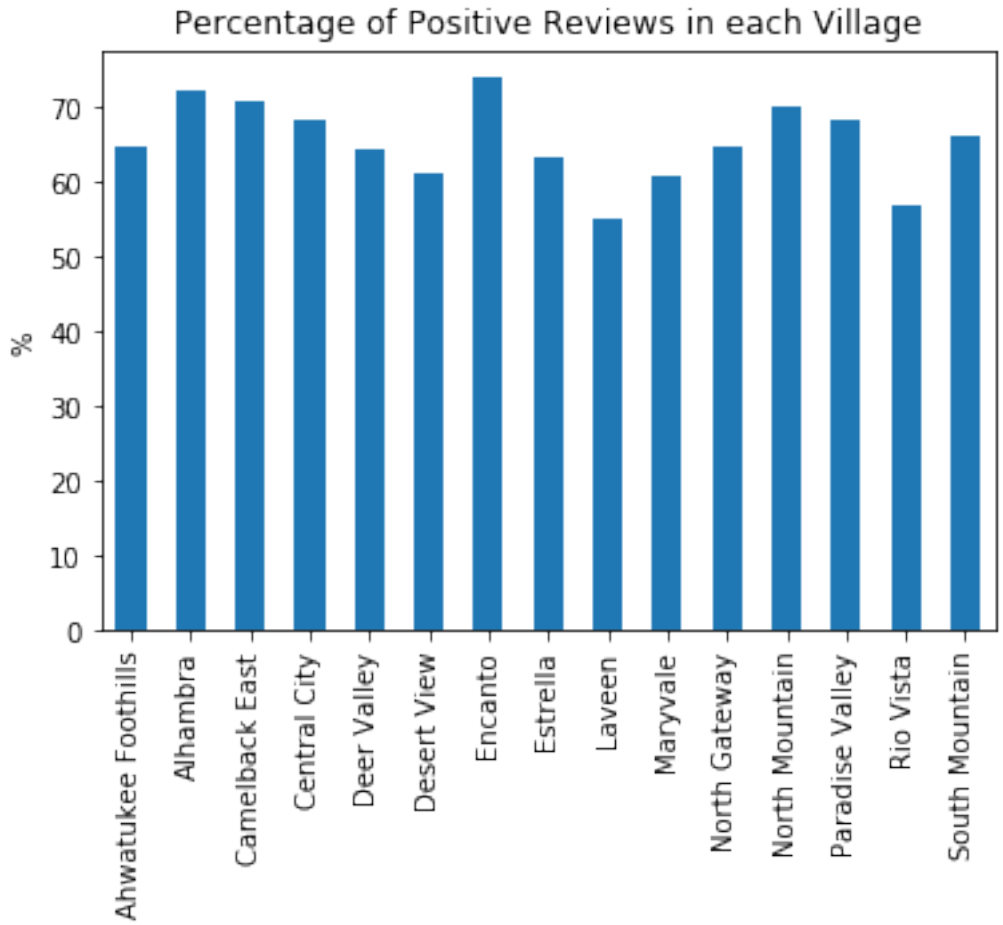
8 Sentiment by Neighborhood

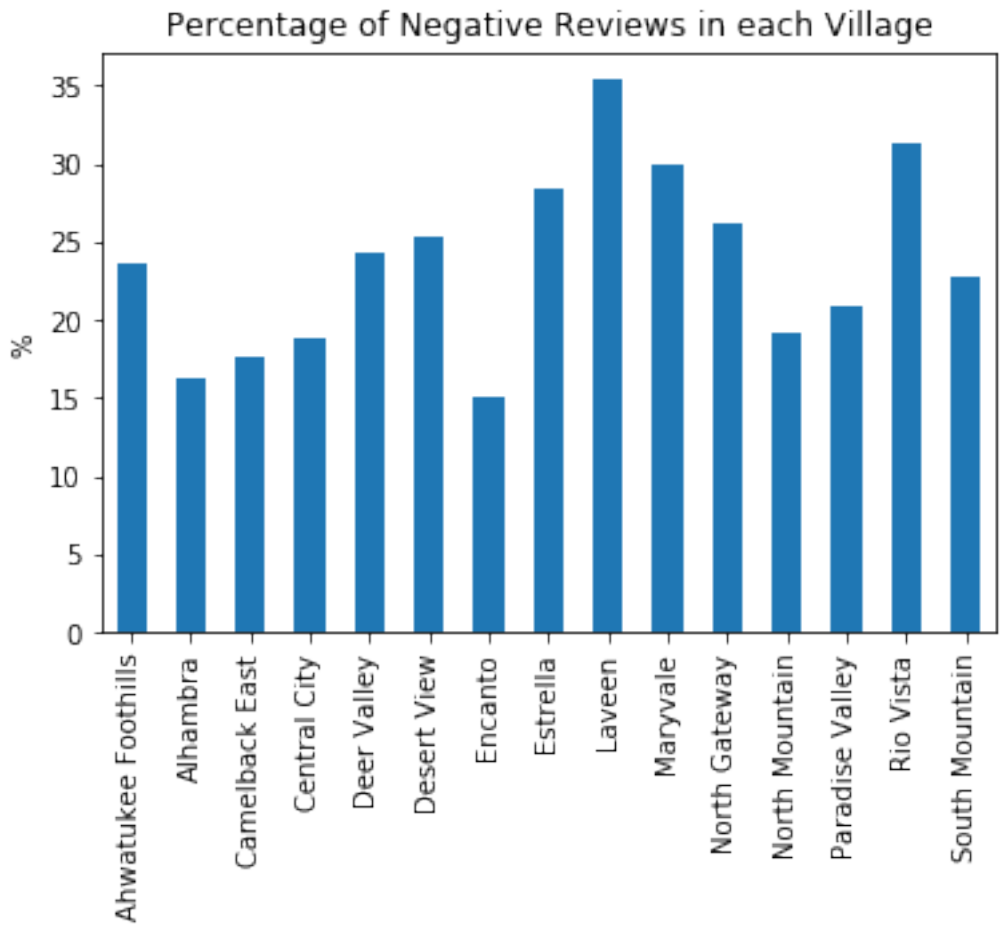
[6]:
```
%%time
def namethevillage(data):
    for village in arizona['NAME']:
        if any(arizona[arizona['NAME'] == village].geometry.contains(data)):
            return village
arizonafoodbusiness['geometry'] = arizonafoodbusiness.apply(
    lambda h: Point(float(h['longitude']), float(h['latitude'])), axis = 1)
arizonafoodbusiness['Village'] = arizonafoodbusiness['geometry'].
 ↪map(namethevillage)
arizonafoodreviews['Village'] = arizonafoodreviews['business_id'].
 ↪map(dict(zip(arizonafoodbusiness['business_id'],␣
 ↪arizonafoodbusiness['Village'])))
```

```
CPU times: user 4min 21s, sys: 14.2 s, total: 4min 35s
Wall time: 6min 34s

//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8:
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
```

[18]:
```
arizonafoodreviews['Village'].value_counts().plot(kind = 'bar')
plt.title("Number of Reviews in each Village")
plt.xticks(rotation = 90)
plt.ylabel("Count")
plt.show();

(arizonafoodreviews[arizonafoodreviews['stars'] > 3]['Village'].value_counts()/
 ↪arizonafoodreviews['Village'].value_counts() * 100).plot(kind = 'bar')
plt.title("Percentage of Positive Reviews in each Village")
plt.xticks(rotation = 90)
plt.ylabel("%")
plt.show();

(arizonafoodreviews[arizonafoodreviews['stars'] < 3]['Village'].value_counts()/
 ↪arizonafoodreviews['Village'].value_counts() * 100).plot(kind = 'bar')
plt.title("Percentage of Negative Reviews in each Village")
plt.xticks(rotation = 90)
plt.ylabel('%')
plt.show();
```




Percentage of Positive Reviews in each Village
%
0
10
20
30
40
50
60
70
Ahwatukee Foothills
Alhambra
Camelback East
Central City
Deer Valley
Desert View
Encanto
Estrella
Laveen
Maryvale
North Gateway
North Mountain
Paradise Valley
Rio Vista
South Mountain



9 FINDINGS

1. Reviews tend to be more positive than negative
2. Negative reviews tend to be about customer service and their experiences at the restaurant.
3. Star rating is not a good predictor of likeability. Review sentiment is a better predictor of likeability.
4. Encanto has the highest percentages of positive reviews.

[ ]:

[ ]: