1 Predicting NYC Restaurant Health Violations

Contents

2 Introduction

Cities across the United States are capitalizing on data. Predictive modeling is becoming a prominent tool for public safety in many cities where the local authority can target interventions. These preventative measures range from predicting which landlords are not complying with city ordinances to which buildings pose the highest risk of fire so they can send inspectors out and prevent an outbreak before it happens.

According to the CDC, more than 48 million Americans per year become sick from food, and an estimated 75% of the outbreaks came from food prepared by caterers, delis, and restaurants. In most cities, health inspections are generally random, which can increase time spent on spot checks at clean restaurants that have been following the rules closely — and missed opportunities to improve health and hygiene at places with more pressing food safety issues.

2.1 Objective

The goal for this project is to leverage public citizen generated data from social media to narrow the search for critical health and safety violations in New York City. As the City of New York manages

an open data portal, everyone can access historical hygiene inspections and violation records. By combine these two data source this project aims to determine which words, phrases, ratings, and patterns among restaurants lead to critical health and safety violations. This model can assist city health inspectors do their job better by prioritizing the kitchens most likely to be in violation of code.

Contents

The New York Health Department inspects the approximately 27,000 restaurants within the city to monitor their compliance with food safety regulations. Inspectors observe how food is prepared, served and stored and whether restaurant workers are practicing good hygiene. They check food temperatures, equipment maintenance and pest control measures.

2.1.1 Imports

In [2]:

```
from IPython import display
from bs4 import BeautifulSoup as bs
import requests
import json
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import geopandas as gpd
from shapely.geometry import Point, Polygon
import plotly.express as px

import warnings
import time

import folium
import folium.plugins as plugins

#misc
import glob, os
import ast

call_apis = False
warnings.filterwarnings("ignore")
pd.set_option('display.max_colwidth',200)
pd.set_option('display.max_columns',50)
pd.options.display.float_format = '{:.2f}'.format

from sklearn.preprocessing import OneHotEncoder

#NLP
import spacy
nlp = spacy.load("en_core_web_lg")
from spacy import displacy
import nltk
import string
from nltk.collocations import *
from nltk import word_tokenize,wordpunct_tokenize
from nltk.tokenize import RegexpTokenizer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from wordcloud import WordCloud
from nltk import FreqDist
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess
from nltk import word_tokenize
#nltk.download('wordnet')

# sk-learn
from sklearn import set_config
set_config(display='diagram')
from sklearn import metrics
from sklearn.metrics import confusion_matrix
```

Contents

```
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTrans
from sklearn.tree import DecisionTreeClassifier
from sklearn.dummy import DummyClassifier
from sklearn.utils.class_weight import compute_class_weight
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score
import imblearn.pipeline
from imblearn.over_sampling import RandomOverSampler

# Deep Learning
from tensorflow import keras
from keras.models import load_model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding
from tensorflow.keras.layers import Dropout, Activation, Bidirectional,
from tensorflow.keras.models import Sequential
from tensorflow.keras import initializers, regularizers, constraints, o
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.preprocessing import text, sequence
from tensorflow.keras.utils import to_categorical
from keras.preprocessing import text,sequence
from keras_preprocessing.sequence import pad_sequences
```

executed in 43.1s, finished 15:01:57 2022-08-12

In [939]:
```
1
```

executed in 8ms, finished 10:07:22 2022-08-16

3 Obtain

This project utilises 3 data sources:

- Historical health and hygiene inspections recorded by New York City Department of Health and Mental Hygiene (DOHMH) public health inspectors
- Yelp API business data and ratings
- Yelp user generateda reviews

Since 2010, New York City has required restaurants to post letter grades that correspond to scores received from sanitary inspections.Since 2010, New York City has required restaurants to post letter grades that correspond to scores received from sanitary inspections.

- "A" grade: 0 to 13 points for sanitary violations
- "B" grade: 14 to 27 points for sanitary violations

- "C" grade: 28 or more points for sanitary violations

3.1 Obtaining Restaurant Inspection Results from NYC Open Data Portal

The dataset can be obtained here

https://data.cityofnewyork.us/Health/DOHMH-New-York-City-Restaurant-Inspection-Results/43nn-pn8j (https://data.cityofnewyork.us/Health/DOHMH-New-York-City-Restaurant-Inspection-Results/43nn-pn8j)

The dataset was downloaded and saved to this repository. Let's load it in and explore its contents.

Detailed descriptions about each column can be found in the Restaurant Inspection Data Dictionary.

In [4]:
```
doh_df = pd.read_csv('data/nyc_open_data/DOHMH_New_York_City_Restaurant
doh_df
```

executed in 2.83s, finished 15:02:00 2022-08-12

Out[4]:

	CAMIS	DBA	BORO	BUILDING	STREET	ZIPCODE	PHONE	DESC
0	50008319	POULETTE	Manhattan	790	9 AVENUE	10019.00	2129569488	
1	41678734	TINY'S DINER	Bronx	3603	RIVERDALE AVENUE	10463.00	7187087600	

The dataset contains 186,227 inspection results. However, when an inspection results in more than one violation, values for associated fields are repeated for each additional violation record. So let's check how many individual restaurants are in the dataset.

3.1.1 Understanding NYC DOHMH Data

In [5]:
```
# How many unique restaurants are in this dataset?
n_unique = doh_df['CAMIS'].nunique()
print(f'There are {n_unique} unique restaurants in the dataset. ')
```

executed in 28ms, finished 15:02:00 2022-08-12

```
There are 19792 unique restaurants in the dataset.
```

Contents

In [6]:
```
# Get more information about the dataset contents
doh_df.info()
```

executed in 1.10s, finished 15:02:01 2022-08-12

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 186227 entries, 0 to 186226
Data columns (total 26 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   CAMIS                  186227 non-null  int64
 1   DBA                    186200 non-null  object
 2   BORO                   186227 non-null  object
 3   BUILDING               185799 non-null  object
 4   STREET                 186227 non-null  object
 5   ZIPCODE                183146 non-null  float64
 6   PHONE                  186214 non-null  object
 7   CUISINE DESCRIPTION    186226 non-null  object
 8   INSPECTION DATE        186227 non-null  object
 9   ACTION                 186227 non-null  object
 10  VIOLATION CODE         185413 non-null  object
 11  VIOLATION DESCRIPTION  185863 non-null  object
 12  CRITICAL FLAG          186227 non-null  object
 13  SCORE                  186227 non-null  int64
```

In [7]:
```
doh_df['ZIPCODE'].astype(str)
```

executed in 224ms, finished 15:02:02 2022-08-12

Out[7]:
```
0         10019.0
1         10463.0
2         10022.0
3         10022.0
4         11368.0
           ...
186222        nan
186223    10019.0
186224    10036.0
186225    10472.0
186226    10461.0
Name: ZIPCODE, Length: 186227, dtype: object
```

In [8]:
```
1 doh_df['Community Board'].astype(str)
```
executed in 207ms, finished 15:02:02 2022-08-12

Out[8]:
```
0         104.0
1         208.0
2         105.0
3         105.0
4         404.0
          ...
186222      nan
186223    105.0
186224    105.0
186225    209.0
186226    210.0
Name: Community Board, Length: 186227, dtype: object
```

In [9]:
```
1 doh_df['Council District'].astype(str)
```
executed in 188ms, finished 15:02:02 2022-08-12

Out[9]:
```
0         3.0
1        11.0
2         4.0
3         4.0
4        21.0
         ...
186222    nan
186223    4.0
186224    4.0
186225   18.0
186226   13.0
Name: Council District, Length: 186227, dtype: object
```

In [10]:
```
1 doh_df['Census Tract'].astype(str)
```
executed in 192ms, finished 15:02:02 2022-08-12

Out[10]:
```
0       13300.0
1       29500.0
2       11202.0
3       11202.0
4       43702.0
         ...
186222      nan
186223  13700.0
186224  11900.0
186225   5001.0
186226  30000.0
Name: Census Tract, Length: 186227, dtype: object
```

Every row has a restaurant ID, address, date, and score. Let's ensure there aren't any duplicated rows.

Contents

In [11]:
```
print(f'There are {doh_df.duplicated(keep=False).sum()} duplicated rows
```
executed in 812ms, finished 15:02:03 2022-08-12

```
There are 22010 duplicated rows.
```

In [12]:
```
# Let's drop these duplucated rows
doh_df.drop_duplicates(keep='first',inplace=True)
```
executed in 894ms, finished 15:02:04 2022-08-12

In [13]:
```
# Confirming duplicates have been removed
doh_df.shape
```
executed in 18ms, finished 15:02:04 2022-08-12

Out[13]: (174725, 26)

Since this project will be leveraging data publicly generated from social media a lookup value will be needed to call the API and join the tables. The Yelp API has an endpoint for Phone Search. This will allow us to pull Yelp business data for each restaurant by proivinging a telephone number. More information can be found in the documentation here. (https://www.yelp.com/developers/documentation/v3/business_search_phone)

In [14]:
```
# Checking that every restaunt has a phone number
missing_num = doh_df['PHONE'].isna().sum()
print(f'There are {missing_num} restaunts missing a telephone number.')
```
executed in 32ms, finished 15:02:04 2022-08-12

```
There are 13 restaunts missing a telephone number.
```

In [15]:
```
# Since only 13 numbers are missing, these rows can be dropped
doh_df.dropna(subset=['PHONE'],inplace=True)
```
executed in 122ms, finished 15:02:04 2022-08-12

In [16]:
```
# Confirming records were dropped
doh_df['PHONE'].isna().sum()
```
executed in 27ms, finished 15:02:04 2022-08-12

Out[16]: 0

In [17]:
```
# How many unique restaurants are remaining?
n_unique = doh_df['CAMIS'].nunique()
print(f'There are {n_unique} unique restaurants remaining in the datase
```
executed in 17ms, finished 15:02:04 2022-08-12

```
There are 19790 unique restaurants remaining in the dataset.
```

Let's explore the date range for this dataset.

Contents

In [18]:
```
doh_df['INSPECTION DATE'] =  pd.to_datetime(doh_df['INSPECTION DATE'])
begin_date = doh_df['INSPECTION DATE'].min()
end_date = doh_df['INSPECTION DATE'].max()
print(f'The data ranges from {begin_date} to {end_date}')
```

executed in 348ms, finished 15:02:05 2022-08-12

```
The data ranges from 2009-05-16 00:00:00 to 2022-03-23 00:00:00
```

Inspections in this dataset range from May 2009 up to March 2022.

Contents

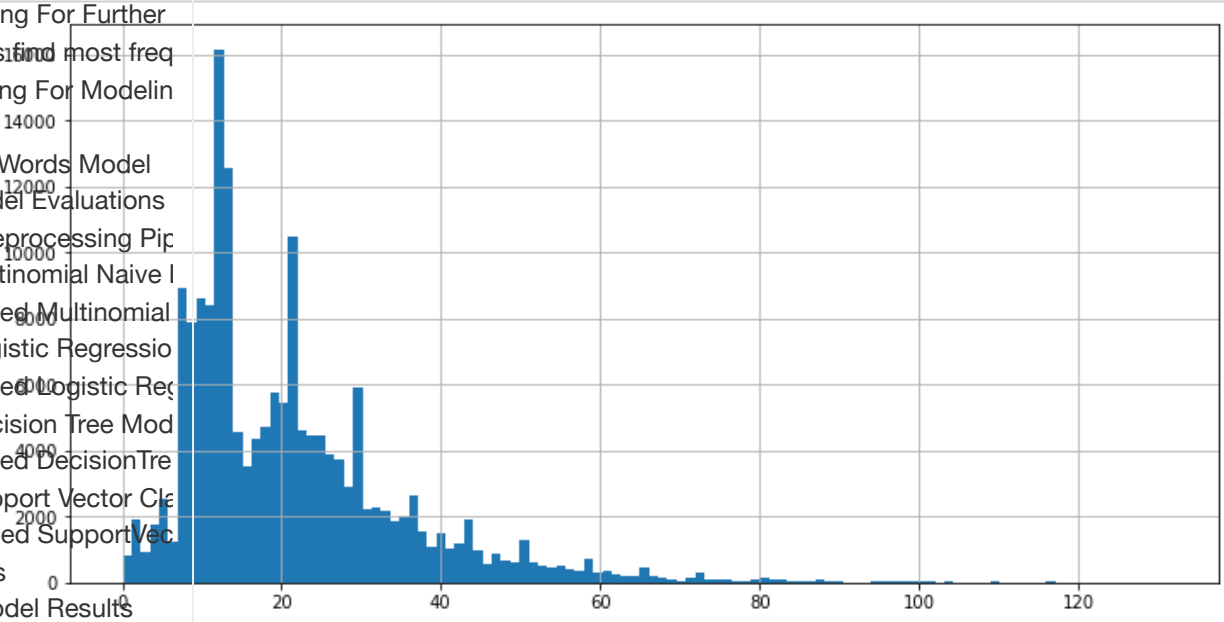
3.1.2 Target Variable -- NYCDOH Inspection Grades

Health code violations found during an inspections carries a point value, and a restaurant's score corresponds to a letter grade. A lower point score, leads to a better letter grade:

- A grade: 0 to 13 points for sanitary violations
- B grade: 14 to 27 points for sanitary violations
- C grade: 28 or more points for sanitary violations

In [19]:
```
# Let see what the score distribution is
doh_df['SCORE'].hist(bins=113, figsize=(12,6));
```

executed in 1.00s, finished 15:02:06 2022-08-12



In [20]:
```
doh_df['SCORE'].describe()
```
executed in 42ms, finished 15:02:06 2022-08-12

Out[20]:
```
count    174712.00
mean         21.36
std          14.36
min           0.00
25%          12.00
50%          18.00
75%          27.00
max         131.00
Name: SCORE, dtype: float64
```

3.1.3 Engineering Target Variable

Because the inspections dataset contains many duplicated businesses in its records, this project will group records by businesses and get a count for each for their letter grades. The target variable will be finding restaurants that have received 1 or more C Grades.

In [21]:
```
# Copying to a new DateFrame
doh_graded = doh_df.copy()
```
executed in 52ms, finished 15:02:06 2022-08-12

In [22]:
```
# Dropping nonrequired features
doh_graded.drop(columns=['INSPECTION DATE', 'ACTION', 'VIOLATION CODE',
                         'VIOLATION DESCRIPTION', 'CRITICAL FLAG','GRADE',
                         'GRADE DATE', 'RECORD DATE', 'INSPECTION TYPE'],inplace=True)
```
executed in 83ms, finished 15:02:06 2022-08-12

In [23]:
```
# Creating new columns with NYCDOH Grades
doh_graded['A'] = (doh_graded['SCORE'] < 14).astype(int)
doh_graded['B'] = (doh_graded['SCORE'] > 13).astype(int) & (doh_df['SCO
doh_graded['C'] = (doh_graded['SCORE'] > 27).astype(int)
```
executed in 40ms, finished 15:02:06 2022-08-12

In [24]:
```
# Creating a new DF and grouping businesses and summing their Grade cou
doh_grouped = doh_graded.groupby(by=['CAMIS', 'DBA','CUISINE DESCRIPTIO
                                     'BORO', 'BUILDING',
                                     'STREET', 'ZIPCODE', 'PHONE', 'Lat
                                     'Longitude', 'Community Board',
                                     'Council District','Census Tract']
```
executed in 2.62s, finished 15:02:09 2022-08-12

In [25]:
```
# Checking the number of businesses that have received 1 or more B Grad
(doh_grouped['B'] > 0).sum()
```
executed in 12ms, finished 15:02:09 2022-08-12

Out[25]:
```
9977
```

Of the 19,790 unique restaurants, 9,977 failed an initial cycle inspection at least once.

In [26]:
```
# Checking the number of businesses that have received 1 or more C Grad
(doh_grouped['C'] > 0).sum()
```
executed in 29ms, finished 15:02:09 2022-08-12

Out[26]: 5648

Contents

Of the 19,790 unique restaurants, 5,648 severly failed an initial cycle inspection at least once and are at risk of being closed by the DOHMH.

In [27]:
```
# Creating the Target Variable 'Severe' for Restaurants that have score
doh_grouped['Severe'] = (doh_grouped['C'] > 0).astype(int)
nyc_df = doh_grouped.reset_index()
nyc_df.drop(['A','B','C'],axis=1, inplace=True)
```
executed in 67ms, finished 15:02:09 2022-08-12

In [28]:
```
# Checking out the new DF with the engineered target variable
nyc_df
```
executed in 38ms, finished 15:02:09 2022-08-12

Out[28]:

	CAMIS	DBA	CUISINE DESCRIPTION	BORO	BUILDING	STREET	ZIPCODE	
0	30075445	MORRIS PARK BAKE SHOP	Bakery Products/Desserts	Bronx	1007	MORRIS PARK AVENUE	10462.00	718
1	30112340	WENDY'S	Hamburgers	Brooklyn	469	FLATBUSH AVENUE	11225.00	718
2	30191841	DJ REYNOLDS PUB AND RESTAURANT	Irish	Manhattan	351	WEST 57 STREET	10019.00	212
3	40356018	RIVIERA CATERERS	American	Brooklyn	2780	STILLWELL AVENUE	11224.00	718
4	40356483	WILKEN'S FINE FOOD	Sandwiches	Brooklyn	7114	AVENUE U	11234.00	718
...	...	...	...	...	...	...	...	
19785	50115169	Sobak	Korean	Manhattan	51	CANAL STREET	10002.00	929
19786	50116155	SEPTEMBER	Coffee/Tea	Brooklyn	83	SARATOGA AVENUE	11233.00	508
19787	50117350	SONG TEA	Coffee/Tea	Manhattan	488	7 AVENUE	10018.00	646
19788	50117434	LA POLLERA COLORADA	Spanish	Queens	8213	NORTHERN BLVD	11372.00	718
19789	50117959	SOL MAYA RESTAURANT	American	Manhattan	2061	2 AVENUE	10029.00	212

19790 rows × 14 columns

NYC DOH Data Exploration

In [29]:
```
1 nyc_df['BORO'].value_counts()
```
executed in 18ms, finished 15:02:09 2022-08-12

Out[29]:
```
Manhattan        7892
Brooklyn         4801
Queens           4468
Bronx            1920
Staten Island     704
0                   5
Name: BORO, dtype: int64
```

In [30]:
```
1 nyc_df['CUISINE DESCRIPTION'].value_counts()
```
executed in 22ms, finished 15:02:09 2022-08-12

Out[30]:
```
American                 4269
Chinese                  1716
Coffee/Tea               1316
Pizza                    1249
Italian                   829
                         ...
Iranian                     3
Fruits/Vegetables           2
Fusion                      1
Chilean                     1
Basque                      1
Name: CUISINE DESCRIPTION, Length: 84, dtype: int64
```

In [31]:
```
1 nyc_df['ZIPCODE'].value_counts()
```
executed in 21ms, finished 15:02:09 2022-08-12

Out[31]:
```
10019.00    513
10036.00    469
10003.00    463
11001.00    367
10022.00    359
           ...
10115.00      1
10173.00      1
10285.00      1
10177.00      1
10069.00      1
Name: ZIPCODE, Length: 221, dtype: int64
```

Contents

3.2 Obtaining Yelp Buniness and Review Data

Now that we have an understanding of the city's inspection results and have explored that dataset it is time to pull in data from the crowd-sourced review platform Yelp.

In [32]:
```
# Loading in locally stored API credentials.
# You can sign up for access and obtain credentials to the Yelp API her
# https://www.yelp.com/developers/documentation/v3

with open('/Users/Rob/.secret/yelp_api.json') as f:
    creds = json.load(f)
```
executed in 18ms, finished 15:02:09 2022-08-12

In [33]:
```
# Checking creds were properly loaded in
creds.keys()
```
executed in 20ms, finished 15:02:09 2022-08-12

Out[33]:
```
dict_keys(['api_key'])
```

3.2.1 Yelp Business Search

In [34]:
```
# Formatting phone numbers provided in the NYCDOH dataset
nyc_df['PHONE'] = '+1'+nyc_df['PHONE']

# Ensure the list contains unique phone numbers only
phone_numbers = set(nyc_df['PHONE'])
phone_numbers = list(phone_numbers)
number_count = len(phone_numbers)
print(f'There are {number_count} unique phone numbers.')
```
executed in 38ms, finished 15:02:09 2022-08-12

```
There are 18937 unique phone numbers.
```

So we will call the Yelp API at the Phone Search Endpoint for all the numbers in the `phone_numbers` list. However the API only allows 5000 callers per day so we'll slice the list into smaller list.

In [35]:
```
# Slicing the phone list into smaller list to fit under the API daily l
# phone_numbers1 = phone_numbers[1:1000]
# phone_numbers2 = phone_numbers[1000:2000]
# phone_numbers3 = phone_numbers[2000:2500]
# phone_numbers4 = phone_numbers[2500:3500]
# phone_numbers5 = phone_numbers[3500:5000]
# phone_numbers6 = phone_numbers[5000:6000]
# phone_numbers7 = phone_numbers[6000:7500]
# phone_numbers8 = phone_numbers[7500:10000]
# phone_numbers9 = phone_numbers[10000:12500]
# phone_numbers11 = phone_numbers[15000:17500]
# phone_numbers12 = phone_numbers[17500:20000]
# phone_numbers9 = phone_numbers[10000:12500]
# phone_numbers10 = phone_numbers[12500:15000]
# phone_numbers11 = phone_numbers[15000:17500]
# phone_numbers12 = phone_numbers[17500:19000]
```
executed in 6ms, finished 15:02:09 2022-08-12

Contents

Contents

In [36]:
```
#Functionizing the Yelp API Phone Search

def get_businesses(phone_numbers):
    """Input a list of formatted phone numbers
    (must start with + and include the country code, like +14159083801)
    and returns a corresponding list of Yelp Businesses"""

    biz_list = []

    for number in phone_numbers:
        url = 'https://api.yelp.com/v3/businesses/search/phone'
        headers = {'Authorization': 'Bearer ' + creds['api_key']}
        url_params = {'phone': number}
        response = requests.get(url, headers=headers, params=url_params
        response_json = response.json()
        biz_list.extend(response_json.get('businesses','U'))

    while 'U' in biz_list:
        biz_list.remove('U')

    return biz_list
```

executed in 11ms, finished 15:02:09 2022-08-12

In [37]:
```
# Call `get_business` function
if call_apis == True:
    biz_list12 = get_businesses(phone_numbers12)

    #Save returned list as a DataFrame and .csv file
    biz12_df = pd.DataFrame(biz_list12)
    biz12_df.to_csv('data/yelp_data/yelp_business/yelp_phone12.csv',ind
```

executed in 8ms, finished 15:02:09 2022-08-12

In [38]:
```
# List of files containing Yelp business data
fpath = 'data/yelp_data/yelp_businesses/'
os.listdir(fpath)
query = fpath+"*.csv"
f_list = glob.glob(query)
```

executed in 10ms, finished 15:02:09 2022-08-12

In [39]:
```
# Append saved Yelp Business tables to a dict
yelp_tables = {}

for f in f_list:
    temp_df = pd.read_csv(f)
    fname = f.replace('data/yelp_data/yelp_businesses/yelp_phone','df_'
    yelp_tables[fname] = temp_df
```

executed in 614ms, finished 15:02:10 2022-08-12

In [40]:
```
yelp_df_list = [t for t in list(yelp_tables.keys())]
```

executed in 7ms, finished 15:02:10 2022-08-12

In [41]:
```
# Concatenating all Yelp Businesses responses from the Phone Search
yelp_businesses_df = pd.concat(yelp_tables,ignore_index=True)
yelp_businesses_df
```

executed in 55ms, finished 15:02:10 2022-08-12

4	rYl_R-UILYqA6vYsbNwLsg	company-new-york-14	Joe Coffee Company	https://s3-media0.fl.yelpc…	MC
...	...	...	...	...	
17297	K_e7_aNgghFLZ2zvjFSNkQ	loreley-beer-garden-new-york-2	Loreley Beer Garden	media0.fl.yelpcdn.com/bphoto/RHzuMs	

Exploring Yelp Businesses Response Data

In [42]:
```
yelp_businesses_df.info()
```

executed in 84ms, finished 15:02:10 2022-08-12

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17302 entries, 0 to 17301
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              17302 non-null  object 
 1   alias           17302 non-null  object 
 2   name            17302 non-null  object 
 3   image_url       16563 non-null  object 
 4   is_closed       17302 non-null  bool   
 5   url             17302 non-null  object 
 6   review_count    17302 non-null  int64  
 7   categories      17302 non-null  object 
 8   rating          17302 non-null  float64
 9   coordinates     17302 non-null  object 
 10  transactions    17302 non-null  object 
 11  price           14501 non-null  object 
 12  location        17302 non-null  object 
 13  phone           17302 non-null  int64  
 14  display_phone   17302 non-null  object 
dtypes: bool(1), float64(1), int64(2), object(11)
memory usage: 1.9+ MB
```

In [43]:
```
1 yelp_businesses_df['price'].value_counts(normalize=True)
```
executed in 15ms, finished 15:02:10 2022-08-12

Out[43]:
```
$$      0.53
$       0.37
$$$     0.07
$$$$    0.02
Name: price, dtype: float64
```

In [44]:
```
1 yelp_businesses_df['rating'].value_counts(normalize=True)
```
executed in 16ms, finished 15:02:10 2022-08-12

Out[44]:
```
4.00    0.30
3.50    0.25
3.00    0.14
4.50    0.12
2.50    0.08
2.00    0.04
5.00    0.03
1.50    0.02
1.00    0.01
Name: rating, dtype: float64
```

In [45]:
```
1 yelp_businesses_df['categories'].value_counts(normalize=True)
```
executed in 26ms, finished 15:02:10 2022-08-12

Out[45]:
```
[{'alias': 'chinese', 'title': 'Chinese'}]
0.05
[{'alias': 'pizza', 'title': 'Pizza'}]
0.04
[{'alias': 'coffee', 'title': 'Coffee & Tea'}]
0.02
[{'alias': 'mexican', 'title': 'Mexican'}]
0.02
[{'alias': 'italian', 'title': 'Italian'}]
0.02
...
[{'alias': 'bakeries', 'title': 'Bakeries'}, {'alias': 'gourmet', 'titl
e': 'Specialty Food'}, {'alias': 'desserts', 'title': 'Desserts'}]   0.00
[{'alias': 'mexican', 'title': 'Mexican'}, {'alias': 'bars', 'title': 'Ba
rs'}, {'alias': 'venues', 'title': 'Venues & Event Spaces'}]       0.00
[{'alias': 'pizza', 'title': 'Pizza'}, {'alias': 'breakfast_brunch', 'tit
le': 'Breakfast & Brunch'}]                                        0.00
[{'alias': 'pubs', 'title': 'Pubs'}, {'alias': 'restaurants', 'title': 'R
estaurants'}]                                                      0.00
[{'alias': 'dimsum', 'title': 'Dim Sum'}, {'alias': 'cafes', 'title': 'Ca
fes'}, {'alias': 'sandwiches', 'title': 'Sandwiches'}]             0.00
Name: categories, Length: 4277, dtype: float64
```

Contents

In [46]:
```
# Duplicates?
yelp_businesses_df[yelp_businesses_df.duplicated(['id'], keep=False)].c
```
executed in 71ms, finished 15:02:10 2022-08-12

Out[46]:
```
id              8474
alias           8474
name            8474
image_url       8123
is_closed       8474
url             8474
review_count    8474
categories      8474
rating          8474
coordinates     8474
transactions    8474
price           7109
location        8474
phone           8474
display_phone   8474
dtype: int64
```


In [47]:
```
yelp_businesses_df.describe()
```
executed in 37ms, finished 15:02:10 2022-08-12

Out[47]:

	review_count	rating	phone
count	17302.00	17302.00	17302.00
mean	189.46	3.53	15227799712.25
std	378.43	0.79	2494930771.02
min	1.00	1.00	12062250663.00
25%	17.00	3.00	12127498940.50
50%	67.00	3.50	17182416042.00
75%	209.00	4.00	17186241444.00
max	13458.00	5.00	19732238527.00

In [48]:
```
yelp_businesses_df['review_count'].sum()
```
executed in 15ms, finished 15:02:10 2022-08-12

Out[48]:
```
3278066
```

3.2.2 Yelp Reviews

Now that all the restaurants from the NYC DOHMH dataset have been used to search the Yelp API and have been concatenated we can use the return url to gather reviews for each business.

In [49]:
```
# OLD from webscraping
# df_10_2 = df_10.loc[1000:2173]
# df_10_2.to_csv('df_10_2',index=False)
```
executed in 12ms, finished 15:02:10 2022-08-12

In [50]:
```
def get_text(url_list):
    """ Given a list of urls, this function will iterate through the li
    extract text from the first page of reviews. The data will be joine
    a corpus for each business"""

    review_txt = []

    for url in url_list:
        req = requests.get(url,allow_redirects=False)
        soup = bs(req.content)
        comments = soup.find_all(class_='raw__09f24__T4Ezm', lang="en")
        comment_txt = []

        for comment in comments:
            comment_txt.append(comment.text)

        comment_corp = ('.'.join(comment_txt))
        review_txt.append(comment_corp)
    return review_txt
```
executed in 12ms, finished 15:02:10 2022-08-12

Due to the time required to run this function, it can be broken into smaller requests. Since we already have smaller list used when we called the API earlier we can take the urls returned from the API here.

In [51]:
```
# Obtaining list of yelp business urls from saved API response
url_list = list(yelp_tables['df_1']['url'])
len(url_list)
```
executed in 10ms, finished 15:02:10 2022-08-12

Out[51]: 859

In [52]:
```
# Calling `get_text` function to obtain Yelp reviews
if call_apis == True:
    review_text = get_text(url_list)
```
executed in 8ms, finished 15:02:10 2022-08-12

In [53]:
```
# Saving the Reviews to a csv in the repository
if call_apis == True:
    rvw_txt = pd.DataFrame(review_text,columns=['Review_Text'])
    rvw_txt.to_csv('rvw_txt1.csv',index=False)
```
executed in 9ms, finished 15:02:10 2022-08-12

Repeat this for all the urls returned from the API

3.2.3 Joining Yelp Reviews to Yelp Business Tables

Contents

In [54]:
```
# List of files containing Yelp business data
fpath = 'data/yelp_data/yelp_reviews/'
os.listdir(fpath)
query = fpath+"*.csv"
f_list = glob.glob(query)
```
executed in 11ms, finished 15:02:10 2022-08-12

Contents

In [55]:
```
# Append saved Yelp Reviews to a dict
rvw_tables = {}

for file in f_list:
    temp_df = pd.read_csv(file)
    fname = file.replace('data/yelp_data/yelp_reviews/rvw_txt','txt_').
    rvw_tables[fname] = temp_df
```
executed in 1.97s, finished 15:02:12 2022-08-12

In [56]:
```
rvw_df_list = [t for t in list(rvw_tables.keys())]
```
executed in 4ms, finished 15:02:12 2022-08-12

In [57]:
```
rvw1 = pd.read_csv('data/yelp_data/yelp_reviews/rvw_txt1.csv')
```
executed in 101ms, finished 15:02:12 2022-08-12

In [58]:
```
yelp_tables.keys()
```
executed in 15ms, finished 15:02:12 2022-08-12

Out[58]:
```
dict_keys(['df_8', 'df_9', 'df_1', 'df_2', 'df_3', 'df_11', 'df_7', 'df_
6', 'df_10', 'df_12', 'df_4', 'df_5'])
```

In [59]:
```
rvw_tables.keys()
```
executed in 16ms, finished 15:02:12 2022-08-12

Out[59]:
```
dict_keys(['txt_11', 'txt_10', 'txt_12', 'txt_2', 'txt_3', 'txt_1', 'txt_
4', 'txt_5', 'txt_7', 'txt_6', 'txt_8', 'txt_9'])
```

In [60]:
```
# Adding review text to the Yelp Business tables
yelp_tables['df_1']['Reviews'] =  rvw_tables['txt_1']
yelp_tables['df_2']['Reviews'] =  rvw_tables['txt_2']
yelp_tables['df_3']['Reviews'] =  rvw_tables['txt_3']
yelp_tables['df_4']['Reviews'] =  rvw_tables['txt_4']
yelp_tables['df_5']['Reviews'] =  rvw_tables['txt_5']
yelp_tables['df_6']['Reviews'] =  rvw_tables['txt_6']
yelp_tables['df_7']['Reviews'] =  rvw_tables['txt_7']
yelp_tables['df_8']['Reviews'] =  rvw_tables['txt_8']
yelp_tables['df_9']['Reviews'] =  rvw_tables['txt_9']
yelp_tables['df_10']['Reviews'] =  rvw_tables['txt_10']
yelp_tables['df_11']['Reviews'] =  rvw_tables['txt_11']
yelp_tables['df_12']['Reviews'] =  rvw_tables['txt_12']
```
executed in 30ms, finished 15:02:12 2022-08-12

In [61]:
```
# Concatenating all Yelp businesses tables with review text included
yelp_df = pd.concat(yelp_tables,ignore_index=True)
yelp_df
```

executed in 54ms, finished 15:02:12 2022-08-12

Contents

2	in7QM4vNUTHVZIi7Outetg	mcdonalds-new-york-430	McDonald's	media0.fl.yelpcdn.com/bphoto/bNmu8D1sj	

In [62]:
```
# OLD Yelp Review API
# review_df = pd.DataFrame(response_json.get('reviews'))
# review_df
```

executed in 5ms, finished 15:02:12 2022-08-12

3.3 Joining NYC DOHMH & Yelp Datasets

In [63]:
```
# Formatting Yelp phone numbers to align with NYC phone numbers to join
yelp_df['phone'] = '+' + yelp_df['phone'].apply(str)
```

executed in 25ms, finished 15:02:12 2022-08-12

In [64]:
```
1 yelp_df.head()
```
executed in 45ms, finished 15:02:13 2022-08-12

Out[64]:

	id	alias	name	
0	RzKR9QcYP03EQ_8DxTxowg	barrio-chino-new-york	Barrio Chino	media0.fl.yelpcdn.com/bphoto/dpaRqPZD6xqI
1	hrP3xLWJ8sLwG_XxEGq61Q	reyes-restaurant-woodside	Reyes Restaurant	https://s3-media0.fl.yelpcdn.com/bphoto/
2	[illegible]NUTHVZIi7Outetg	mcdonalds-new-york-430	McDonald's	media0.fl.yelpcdn.com/bphoto/bNmu8D1sjwJzb

Contents

	id	alias	name	
3	F-YGKegXRG2TE-Xw1AWeCA	popeyes-louisiana-kitchen-rockaway-beach-2	Popeyes Louisiana Kitchen	media0.fl.yelpcdn.com/bphoto/A4MR2hDN3mWef
4	YYL_R-UlLYqA6vYsbNwLsg	joe-coffee-company-new-york-14	Joe Coffee Company	https://s3-media0.fl.yelpcdn.co MQAN4

Contents

In [65]:
```
# Merging NYC and Yelp datasets
df_1 = pd.merge(nyc_df, yelp_df, left_on='PHONE', right_on='phone', how
```

executed in 71ms, finished 15:02:13 2022-08-12

In [66]:
```
# Saving merged dataset to csv
# df_1.to_csv('full_dataset.csv')
df_1
```

executed in 59ms, finished 15:02:13 2022-08-12

Out[66]:

Contents

	CAMIS	DBA	CUISINE DESCRIPTION	BORO	BUILDING	STREET	ZIPCODE	
0	30112340	WENDY'S	Hamburgers	Brooklyn	469	FLATBUSH AVENUE	11225.00	+17182
		RIVIERA				STILLWELL		

In [67]:
```
1 df_1.info()
```
executed in 144ms, finished 15:02:13 2022-08-12

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18462 entries, 0 to 18461
Data columns (total 30 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   CAMIS                 18462 non-null  int64  
 1   DBA                   18461 non-null  object 
 2   CUISINE DESCRIPTION   18462 non-null  object 
 3   BORO                  18462 non-null  object 
 4   BUILDING              18302 non-null  object 
 5   STREET                18462 non-null  object 
 6   ZIPCODE               18168 non-null  float64
 7   PHONE                 18462 non-null  object 
 8   Latitude              18420 non-null  float64
 9   Longitude             18420 non-null  float64
 10  Community Board       17992 non-null  float64
 11  Council District      17992 non-null  float64
 12  Census Tract          17992 non-null  float64
 13  Severe                18462 non-null  int64  
 14  id                    18462 non-null  object 
 15  alias                 18462 non-null  object 
 16  name                  18462 non-null  object 
 17  image_url             17672 non-null  object 
 18  is_closed             18462 non-null  bool   
 19  url                   18462 non-null  object 
 20  review_count          18462 non-null  int64  
 21  categories            18462 non-null  object 
 22  rating                18462 non-null  float64
 23  coordinates           18462 non-null  object 
 24  transactions          18462 non-null  object 
 25  price                 15391 non-null  object 
 26  location              18462 non-null  object 
 27  phone                 18462 non-null  object 
 28  display_phone         18462 non-null  object 
 29  Reviews               15681 non-null  object 
dtypes: bool(1), float64(7), int64(3), object(19)
memory usage: 4.2+ MB
```

Contents

4 Scrubbing The Data

In [68]:
```
1 # Drop businesses missing reviews
2 # Drop unneccessary columns
3 # Check duplicates
4 # !Feature Engineering
5 # Get coordinates
6 # Get takeout/delivery
7 # Get $$$$
```
executed in 7ms, finished 15:02:13 2022-08-12

4.1 Drop businesses missing reviews and

duplicates

In [69]:
```
df_1['Reviews'].isna().sum()
```
executed in 18ms, finished 15:02:13 2022-08-12

Out[69]: 2781

In [70]:
```
df_1.dropna(subset='Reviews',inplace=True)
df_1['Reviews'].isna().sum()
```
executed in 78ms, finished 15:02:13 2022-08-12

Out[70]: 0

In [71]:
```
# Checking for duplicated rows
df_1[df_1.duplicated(keep='first')]
```
executed in 583ms, finished 15:02:14 2022-08-12

Out[71]:

	CAMIS	DBA	CUISINE DESCRIPTION	BORO	BUILDING	STREET	ZIPCODE	PHO
12	40362264	P & S DELI GROCERY	American	Manhattan	730	COLUMBUS AVENUE	10025.00	+12129323

In [72]:
```
# Dropping Duplicated Rows
df_1.drop_duplicates(keep='first',inplace=True)
```
executed in 376ms, finished 15:02:14 2022-08-12

Contents

In [73]:
```
1 df_1
```
executed in 55ms, finished 15:02:14 2022-08-12

Contents

	camis	dba	cuisine_description	boro	building	street	zipcode	phone
1	40356018	RIVIERA CATERERS	American	Brooklyn	2780	STILLWELL AVENUE	11224.00	+17183

In [74]:
```
1 #Resetting the index
2 df_1.reset_index(inplace=True)
3 df_1.drop(columns='index',inplace=True)
```
executed in 30ms, finished 15:02:14 2022-08-12

In [75]:
```
1 df_1.info()
```
executed in 95ms, finished 15:02:14 2022-08-12

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13061 entries, 0 to 13060
Data columns (total 30 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   CAMIS                 13061 non-null  int64  
 1   DBA                   13060 non-null  object 
 2   CUISINE DESCRIPTION   13061 non-null  object 
 3   BORO                  13061 non-null  object 
 4   BUILDING              12976 non-null  object 
 5   STREET                13061 non-null  object 
 6   ZIPCODE               12854 non-null  float64
 7   PHONE                 13061 non-null  object 
 8   Latitude              13035 non-null  float64
 9   Longitude             13035 non-null  float64
 10  Community Board       12759 non-null  float64
 11  Council District      12759 non-null  float64
 12  Census Tract          12759 non-null  float64
 13  Severe                13061 non-null  int64  
 14  id                    13061 non-null  object 
 15  alias                 13061 non-null  object 
 16  name                  13061 non-null  object 
 17  image_url             12526 non-null  object 
 18  is_closed             13061 non-null  bool   
 19  url                   13061 non-null  object 
 20  review_count          13061 non-null  int64  
 21  categories            13061 non-null  object 
 22  rating                13061 non-null  float64
 23  coordinates           13061 non-null  object 
 24  transactions          13061 non-null  object 
 25  price                 10965 non-null  object 
 26  location              13061 non-null  object 
 27  phone                 13061 non-null  object 
 28  display_phone         13061 non-null  object 
 29  Reviews               13061 non-null  object 
dtypes: bool(1), float64(7), int64(3), object(19)
memory usage: 2.9+ MB
```

Contents

In [76]:
```
1 df_1[df_1['DBA'].isna()]
```
executed in 50ms, finished 15:02:14 2022-08-12

Out[76]:

	CAMIS	DBA	CUISINE DESCRIPTION	BORO	BUILDING	STREET	ZIPCODE	PHONE	Lat
5435	41650149	NaN	Italian	Manhattan	345	EAST 83 STREET	10028.00	+12127378312	

Contents

In [77]:
```
1 df_1.iloc[5435,1] = '83 1/2'
```
executed in 16ms, finished 15:02:14 2022-08-12

In [78]:
```
1 # Looking at the distributions for each Boro
2 df_1['BORO'].value_counts()
```
executed in 94ms, finished 15:02:14 2022-08-12

Out[78]:
```
Manhattan        5746
Queens           2954
Brooklyn         2941
Bronx             976
Staten Island     441
0                   3
Name: BORO, dtype: int64
```

In [79]:
```
# Inspecting missing Boro's
df_1[df_1['BORO'] == '0' ]
```
executed in 48ms, finished 15:02:14 2022-08-12

Out[79]:

	CAMIS	DBA	CUISINE DESCRIPTION	BORO	BUILDING	STREET	ZIPCODE	PHONE
6780	50005134	KINFOLK	American	0	94	WYTHE AVE	11249.00	+1347799294(
6781	50005134	KINFOLK	American	0	94	WYTHE AVE	11249.00	+1347799294(
10182	50060598	POKEWORKS (Entrance is on 41st street between Lexington & Park)	Hawaiian	0	122	E 42ND ST	10168.00	+1646653979:

Contents

In [80]:
```
# Imputing missing BORO labels
df_1.iloc[6780,3] = 'Brooklyn'
df_1.iloc[6781,3] = 'Brooklyn'
df_1.iloc[10182,3] = 'Manhattan'
```
executed in 9ms, finished 15:02:14 2022-08-12

In [81]:
```
# Looking at the distributions for each Boro
df_1['BORO'].value_counts()
```
executed in 21ms, finished 15:02:14 2022-08-12

Out[81]:
```
Manhattan        5747
Queens           2954
Brooklyn         2943
Bronx             976
Staten Island     441
Name: BORO, dtype: int64
```

Contents

In [82]:
```
# Looking at Restaurants with missing zipcodes
df_1[df_1['ZIPCODE'].isna()]
```
executed in 54ms, finished 15:02:14 2022-08-12

287	40387418	TAL BAGELS DELI	Bagels/Pretzels	Manhattan	975979	FIRST AVENUE	NaN	+1

In [83]:
```
# Converting location data from str to dict dtype
df_1['location'] = df_1['location'].apply(lambda x: ast.literal_eval(x))
```
executed in 1.59s, finished 15:02:16 2022-08-12

In [84]:
```
# Confirming change
type(df_1.iloc[0]['location'])
```
executed in 17ms, finished 15:02:16 2022-08-12

Out[84]:
```
dict
```

In [85]:
```
# Imputing Zipcodes missing from NYC data with Yelp value
addrss_list = list(df_1['location'])
zip_list =[]

for zipcode in addrss_list:
    zip_list.append(zipcode['zip_code'])

df_1['zipcode'] = zip_list
df_1.drop(columns=['ZIPCODE'],inplace=True)
```

executed in 60ms, finished 15:02:16 2022-08-12

In [86]:
```
# Converting all review text into string objects
df_1['Reviews'].astype(str,errors='raise')
```

executed in 19ms, finished 15:02:16 2022-08-12

Out[86]:
```
0        Forgot my root beer and the burger tasted very weird.If I can gi
ve this place no stars I absolutely would. The workers here are in effici
ent.  They do not work the best possible way. They cause th...
1        I had my wedding here a month ago and I would give Riviera 10 st
ars if I could. It was such a joy and ease to work with Adam, Tommy, and
their staff.  A true diamond in the rough of Coney Island a...
2        We've recently discovered this gem on Uber Eats and they have th
e best sandwiches!!The broccoli rabe fried eggplant mozzarella sandwich w
as amazing. Having a fresh green veggie paired with a light...
3        We've ordered from these folks a number of times on Grubhub and
have always been extremely satisfied. We've never been to the actual stor
e but I have called over there a few times with some questi...
4        We've ordered from these folks a number of times on Grubhub and
have always been extremely satisfied. We've never been to the actual stor
e but I have called over there a few times with some questi...
                                ...
13056    This is my go to dessert place since its nearby and affordable.
The ice-cream is extremely good, it tastes fresh and seems to be made wit
h the fresh ingredients as well. The coconut and green tea ...
13057    My family wanted bo zai fan and decided to come try here for out
door dining on 10/3. There was 5 of us (plus baby stroller) sitting outsi
de under the tent infront of the shop (indoor dining had ju...
13058    GREATEST PRICE FOR DIMSUM. You don't find prices like this anymo
re, not in NYC, not in Bay Area, not in Canada... It's $3.50 per dish (us
ually), and is pretty darn good. It hits the spot without b...
13059    I really like the drinks here that come with mochi.  It's a laye
r of gooey sticky ricecake that's really fun to drink through a straw, as
opposed to individual pieces of mochi.Also the drinks have...
13060    Because I've been going here since I was 7 years old And never l
et me down some of the best food in myOpinion.Service is good but food is
average and EXPENSIVE. We arrived around 9am and ordered f...
Name: Reviews, Length: 13061, dtype: object
```

Contents

5 Exploring The Dataset

In [558]:
```
df_1['Severe'].value_counts()
```
executed in 318ms, finished 12:23:11 2022-08-16

Out[558]:
```
0    9158
1    3903
Name: Severe, dtype: int64
```

Contents

In [87]:
```
# Dropping unneccessary columns
eda_df = df_1.drop(columns=['display_phone','phone','location','coordin
                            'categories','is_closed','url','image_url','name','ali
```
executed in 13ms, finished 15:02:16 2022-08-12

In [88]:
```
eda_df['price'].value_counts()
```
executed in 18ms, finished 15:02:16 2022-08-12

Out[88]:
```
$      5952
$$     3945
$$$     817
$$$$    251
Name: price, dtype: int64
```

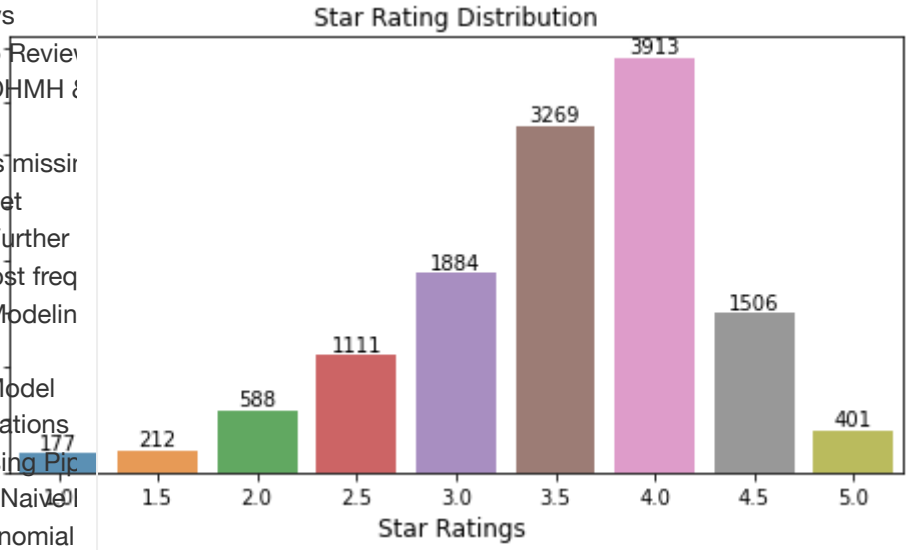
In [89]:
```
eda_df['rating'].value_counts()
```
executed in 18ms, finished 15:02:16 2022-08-12

Out[89]:
```
4.00    3913
3.50    3269
3.00    1884
4.50    1506
2.50    1111
2.00     588
5.00     401
1.50     212
1.00     177
Name: rating, dtype: int64
```

Contents

In [90]:
```
#Get the distribution of the ratings
x=eda_df['rating'].value_counts()
x=x.sort_index()
#plot
plt.figure(figsize=(8,4))
ax= sns.barplot(x.index, x.values, alpha=0.8)
plt.title("Star Rating Distribution")
plt.ylabel('# of businesses', fontsize=12)
plt.xlabel('Star Ratings ', fontsize=12)

#adding the text labels
rects = ax.patches
labels = x.values
for rect, label in zip(rects, labels):
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width()/2, height + 5, label, ha='c

plt.show()
```

executed in 527ms, finished 15:02:17 2022-08-12



In [91]:
```
fig = plt.figure()
```

executed in 17ms, finished 15:02:17 2022-08-12

```
<Figure size 432x288 with 0 Axes>
```

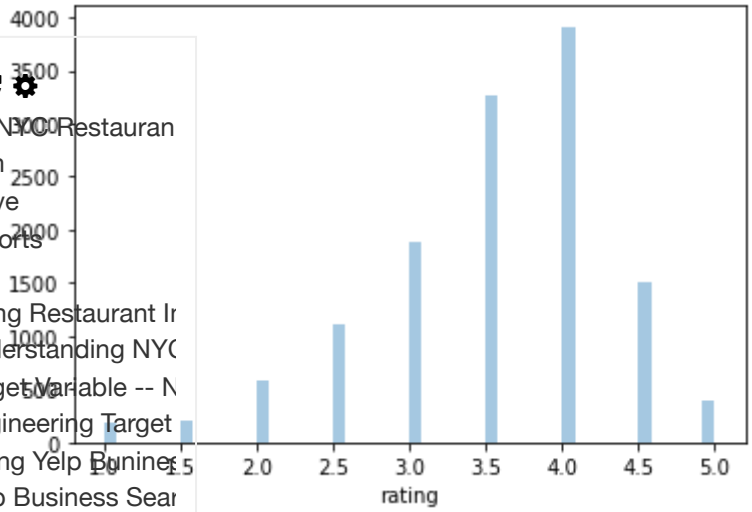
In [92]:
```
eda_df['rating'].value_counts()
```

executed in 15ms, finished 15:02:17 2022-08-12

Out[92]:
```
4.00    3913
3.50    3269
3.00    1884
4.50    1506
2.50    1111
2.00     588
5.00     401
1.50     212
1.00     177
Name: rating, dtype: int64
```

In [93]:
```
1 sns.distplot(eda_df.rating, kde=False)
```
executed in 446ms, finished 15:02:17 2022-08-12

Out[93]: <AxesSubplot:xlabel='rating'>



Contents

In [94]:
```
1 cuisines = pd.DataFrame(eda_df['CUISINE DESCRIPTION'].value_counts())
2 cuisines.reset_index(inplace=True)
3 cuisines[0:10]
```
executed in 22ms, finished 15:02:17 2022-08-12

Out[94]:

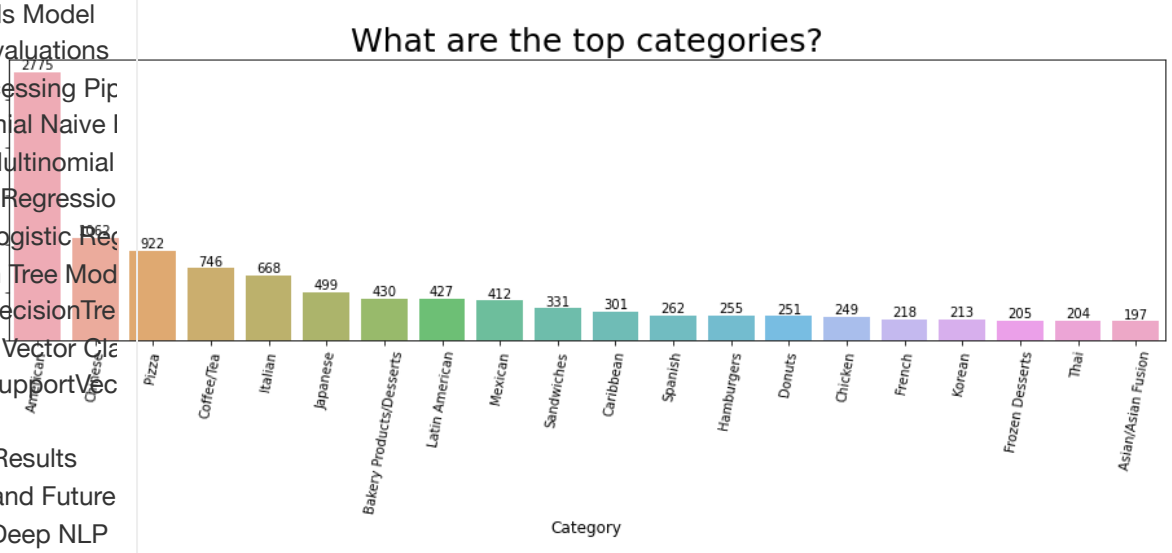
	index	CUISINE DESCRIPTION
0	American	2775
1	Chinese	1062
2	Pizza	922
3	Coffee/Tea	746
4	Italian	668
5	Japanese	499
6	Bakery Products/Desserts	430
7	Latin American	427
8	Mexican	412
9	Sandwiches	331

In [95]:

```
# What are the popular Cuisine categories?
cuisine_cats = eda_df['CUISINE DESCRIPTION']

x = cuisine_cats.value_counts()

print("There are ",len(x)," different types of cuisines in NYC")

#prep for chart
x = x.sort_values(ascending=False)
x = x.iloc[0:20]

#chart
plt.figure(figsize=(16,4))
ax = sns.barplot(x.index, x.values, alpha=0.8)#,color=color[5])
plt.title("What are the top categories?",fontsize=25)
locs, labels = plt.xticks()
plt.setp(labels, rotation=80)
plt.ylabel('# businesses', fontsize=12)
plt.xlabel('Category', fontsize=12)

#adding the text labels
rects = ax.patches
labels = x.values
for rect, label in zip(rects, labels):
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width()/2, height + 5, label, ha='c

plt.show()
```

executed in 973ms, finished 15:02:18 2022-08-12

```
There are  82  different types of cuisines in NYC
```



Contents

In [96]:
```
1 eda_df.columns
```

executed in 12ms, finished 15:02:18 2022-08-12

Out[96]:
```
Index(['CAMIS', 'DBA', 'CUISINE DESCRIPTION', 'BORO', 'BUILDING', 'STREET',
       'PHONE', 'Latitude', 'Longitude', 'Community Board', 'Council District',
       'Census Tract', 'Severe', 'id', 'review_count', 'rating',
       'transactions', 'price', 'Reviews', 'zipcode'],
      dtype='object')
```

Contents

In [97]:
```
# Most reviewed restaurants
eda_df[['CAMIS','id','DBA', 'review_count', 'rating']].sort_values(asce
```

executed in 47ms, finished 15:02:18 2022-08-12

Out[97]:

	CAMIS	id	DBA	review_count	rating
1108	50062888	V7lXZKBDzScDeGB8JmnzSA	KATZS DELICATESSEN (DEKALB MARKET)	13458	4.00
1106	40732665	V7lXZKBDzScDeGB8JmnzSA	KATZ'S DELICATESSEN	13458	4.00
6845	50005848	44SY464xDHbvOcjDzRbKkQ	IPPUDO	10219	4.00
559	40402172	WIhm0W9197f_rRtDziq5qQ	LOMBARDI'S	6289	4.00
7658	50017243	UA2M9QFZghe-9th2KwLoWQ	BURGER & LOBSTER	5592	4.00
5036	41612461	ga6sRtE0l85iftw_5-W84Q	DOMINIQUE ANSEL BAKERY	4971	4.00
2308	41172734	U5hCNNyJmb7f3dmC1HTzSQ	JUNIOR'S RESTAURANT	4971	4.00
2306	41172734	U5hCNNyJmb7f3dmC1HTzSQ	JUNIOR'S RESTAURANT	4970	4.00
5034	41612461	ga6sRtE0l85iftw_5-W84Q	DOMINIQUE ANSEL BAKERY	4969	4.00
5155	41627819	xt4sa64WOrpJvZBDPNPNYg	JACOB'S PICKLES	4765	4.00
3007	41293936	HfJhenvKDi-ds8SUJFwrLg	MOMOFUKU NOODLE BAR	4330	3.50
2250	41159801	nI1UYDCYUTt23TpGxqnLKg	BUDDAKAN	4259	4.00
10242	50061233	vfYhEpp0x-DrNjC6GSJjPQ	ARTICHOKE BASILLE'S PIZZA	4062	4.00
9625	50055243	B3_K2kUVbYOU0VaLcj_LTw	THAI VILLA	4051	4.50
12978	50190033	kBZggrnSP1kcUMnsnfkTaQ	NAN XIANG XIAO LONG BAO	4034	4.00
448	40396464	c3eMI4_o4dPDDhPV_ibBYQ	ESS-A-BAGEL	3950	4.00
2408	41195733	V6xlMNLFTsxB4W0VVcRK_Q	230 FIFTH	3943	3.00
2409	41195733	V6xlMNLFTsxB4W0VVcRK_Q	230 FIFTH	3942	3.00
11060	50071474	j1S3NUrkB3BVT49n_e76NQ	BEST BAGEL & COFFEE	3856	4.50
8911	50045841	TN4RnyqHMSupRFot4Q-_EA	IPPUDO NY	3843	4.00
6311	50000923	C8j0q4Ma_S5hBGuAI-aaww	DI FARA PIZZERIA	3739	4.00
4561	41543887	pfmAcS-g6SiNG0KlLvrqnA	BEAUTY AND ESSEX	3641	4.00
4554	41543410	fxGpXRxFUDzlU3Cyszu4uQ	RED ROOSTER HARLEM, GINNY'S SUPPER CLUB	3499	3.50
9910	50058139	uoT2_Wrt1noD6kZOID7tUg	JANE	3330	4.00
7641	50017190	lWOkeS-wV4no8qqA9OwwEg	DOUGHNUT PLANT	3312	4.50
7647	50079524	lWOkeS-wV4no8qqA9OwwEg	DOUGHNUT PLANT	3312	4.50
8689	50043740	QVWmn5jmy_TPmg0G71Dk5g	SEA THAI	3290	3.50
996	40685858	b8a-8u_A51v2IzyjLVsx6w	DEL FRISCO'S STEAKHOUSE	3209	4.00
4013	41453428	ikazsJps1k-Br2FbunwCtg	SPOT	3200	4.00
3193	50012117	FhXjAc6nKLf414KxYujUHw	ROBERTA'S PIZZA & BAKERY	3077	4.00
3191	41307383	FhXjAc6nKLf414KxYujUHw	ROBERTA'S	3077	4.00

Contents

	CAMIS	id	DBA	review_count	rating
3192	50001089	FhXjAc6nKLf414KxYujUHw	BLANCA	3077	4.00
3407	41351524	A_YpTLbAlEqeLVSs9bxbEA	CLUB A STEAKHOUSE	3076	4.50
2259	41162796	dMhRafXdr765DHe0k-QfaQ	ABC KITCHEN	3072	4.00
5564	41664978	veq1Bl1DW3UWMekZJUsG1Q	GRAMERCY TAVERN	3072	4.50
5563	41664978	veq1Bl1DW3UWMekZJUsG1Q	GRAMERCY TAVERN	3071	4.50
404	40395055	QObHX0yR6zd0WfksRDbJTA	BECCO	2936	3.50
405	40395055	QObHX0yR6zd0WfksRDbJTA	BECCO	2936	3.50
5095	41620680	C-8mGN7lt5rIraO5n-15ug	THE SMITH	2871	4.00
5094	41620680	C-8mGN7lt5rIraO5n-15ug	THE SMITH	2870	4.00
252	40384528	CVxz9B5ncpMwVK1bk_c1YA	VESELKA RESTAURANT	2805	4.00
250	40384528	CVxz9B5ncpMwVK1bk_c1YA	VESELKA RESTAURANT	2803	4.00
5040	41614161	tsiC3VpO-hZEb98Qk3p-DQ	AMY RUTH'S RESTAURANT	2793	4.00
164	40373656	pQuZDnefX038xurT1BhDXA	KEENS STEAKHOUSE	2791	4.00
765	40575863	uc5qQMzs96rzjK27epDCug	FAMOUS JOE'S PIZZA	2780	4.00
5351	41644048	ETgJqJHV7BW6pIr9Ox74sA	AMELIE	2779	4.50
2957	41280221	ogCC-lJJYnwXDvKGmKZ6Sw	WOORIJIP	2726	4.00
1214	40795021	TU_BU9HLflYI2xlyqVQdRA	TAO RESTAURANT	2713	3.50
1215	40795021	TU_BU9HLflYI2xlyqVQdRA	TAO RESTAURANT	2713	3.50
4356	41508109	MhceeLrM2_3mcKw5EGLY6A	PURE THAI RESTAURANT	2684	4.00

Contents

In [98]:
```
# Mapping prices to ordinal categories
price_dict={'$$$$': 4, '$$$': 3, '$$': 2, '$': 1}

eda_df['prices'] = eda_df['price'].map(price_dict)
```

executed in 27ms, finished 15:02:18 2022-08-12

In [99]:
```
#Imputing a 1 for NAN values
eda_df['prices'].fillna(1,inplace=True)
```

executed in 10ms, finished 15:02:18 2022-08-12

In [100]:
```
transactions = eda_df['transactions']
```

executed in 14ms, finished 15:02:18 2022-08-12

In [101]:
```
#Use ast module to convert string objects into list
transactions = transactions.apply(lambda x: ast.literal_eval(x))
```

executed in 295ms, finished 15:02:19 2022-08-12

In [102]:
```
# Change list into dicts
dcts = transactions.apply(lambda x: {c: 1 for c in x})

# Create new dataframe based on the list of dictionaries.
ohe_df = pd.DataFrame(dcts.tolist()).fillna(0)

```

executed in 52ms, finished 15:02:19 2022-08-12

Contents

In [103]:
```
ohe_df
```

executed in 24ms, finished 15:02:19 2022-08-12

Out[103]:

	delivery	pickup	restaurant_reservation
0	1.00	0.00	0.00
1	0.00	0.00	0.00
2	1.00	1.00	0.00
3	1.00	1.00	0.00
4	1.00	1.00	0.00
...	...	...	...
13056	0.00	0.00	0.00
13057	1.00	0.00	0.00
13058	0.00	0.00	1.00
13059	1.00	1.00	0.00
13060	1.00	1.00	0.00

13061 rows × 3 columns

In [104]:
```
# Concatenate dummy variales with the full dataset
eda_df = pd.concat([eda_df,ohe_df],axis=1)
eda_df.head()
```

executed in 39ms, finished 15:02:19 2022-08-12

Out[104]:

	CAMIS	DBA	CUISINE DESCRIPTION	BORO	BUILDING	STREET	PHONE	Latitude
0	30112340	WENDY'S	Hamburgers	Brooklyn	469	FLATBUSH AVENUE	+17182875005	40.66
1	40356018	RIVIERA CATERERS	American	Brooklyn	2780	STILLWELL AVENUE	+17183723031	40.58
2	40356483	WILKEN'S FINE FOOD	Sandwiches	Brooklyn	7114	AVENUE U	+17184443838	40.62
3	40356731	TASTE THE TROPICS ICE CREAM	Frozen Desserts	Brooklyn	1839	NOSTRAND AVENUE	+17188560821	40.64

Contents

	CAMIS	DBA	CUISINE DESCRIPTION	BORO	BUILDING	STREET	PHONE	Latitude
4	40356731	TASTE THE TROPICS ICE CREAM	Frozen Desserts	Brooklyn	1839	NOSTRAND AVENUE	+17188560821	40.64

Contents

In [105]:
```
eda_df.drop(columns=['price','transactions'],inplace=True)
```
executed in 17ms, finished 15:02:19 2022-08-12

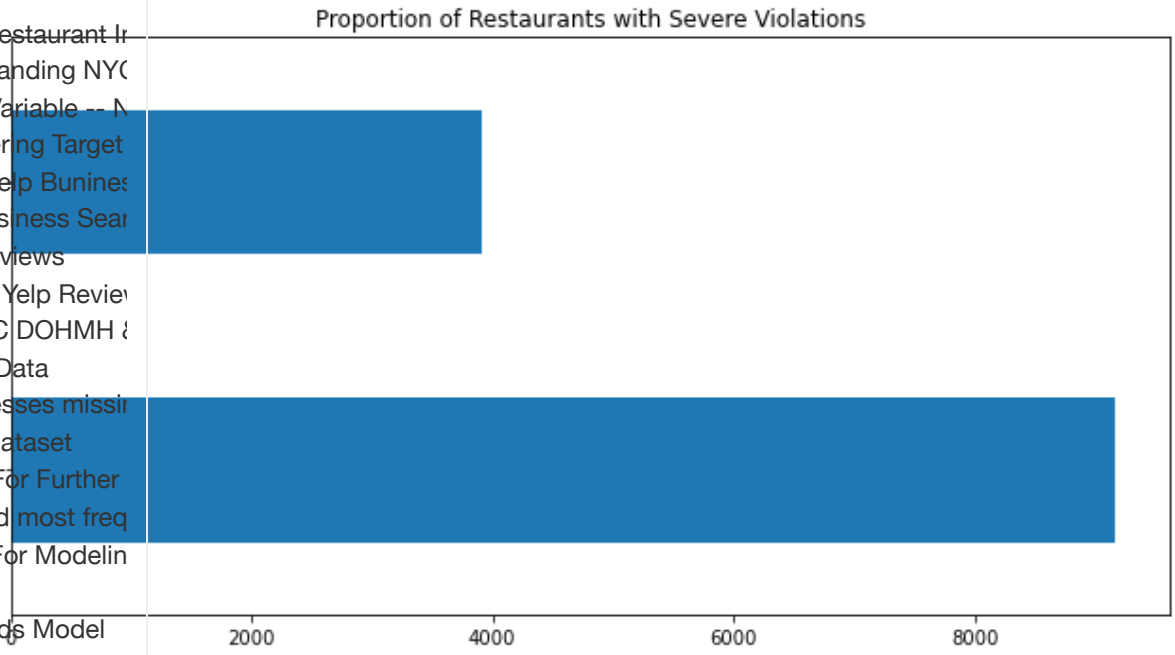
In [106]:
```
# Look at all the feautres compared to the 0 class and 1 class
```
executed in 5ms, finished 15:02:19 2022-08-12

In [107]:
```
labels_df = eda_df['Severe']
labels_df.shape
```
executed in 10ms, finished 15:02:19 2022-08-12

Out[107]:
```
(13061,)
```

In [108]:
```
fig, ax = plt.subplots(figsize=(9.2, 5))

n_obs = labels_df.shape

(eda_df['Severe']
    .value_counts()
    .plot.barh(title="Proportion of Restaurants with Severe Violations")
)
ax.set_ylabel("Severe Violations")

fig.tight_layout()
```

executed in 303ms, finished 15:02:19 2022-08-12



In [109]:
```
boro_counts = (eda_df[['BORO', 'Severe']]
               .groupby(['BORO', 'Severe'])
               .size()
               .unstack('Severe')
)
boro_counts
```

executed in 27ms, finished 15:02:19 2022-08-12

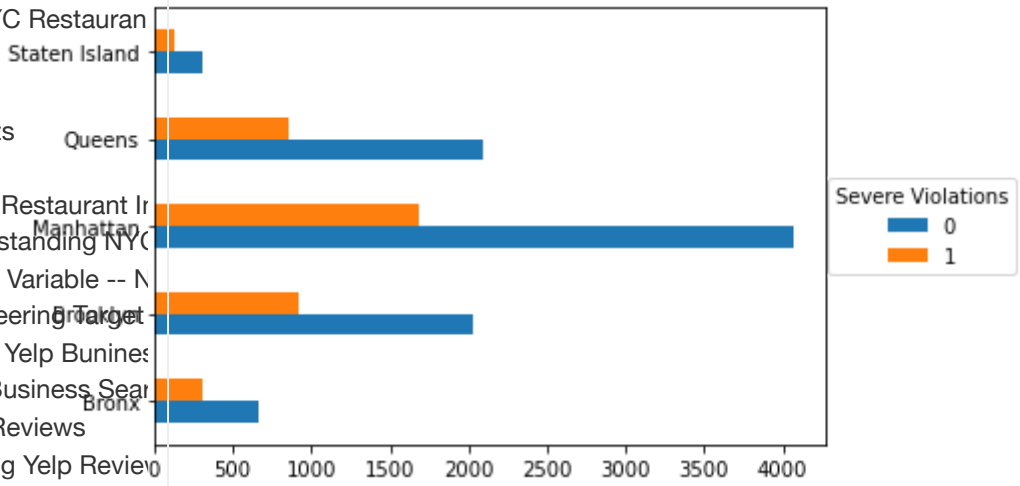
Out[109]:

Severe	0	1
BORO		
Bronx	664	312
Brooklyn	2025	918
Manhattan	4064	1683
Queens	2092	862
Staten Island	313	128

Contents

In [110]:
```
ax = boro_counts.plot.barh()
ax.legend(
    loc='center right',
    bbox_to_anchor=(1.3, 0.5),
    title='Severe Violations'
);
```

executed in 393ms, finished 15:02:20 2022-08-12



Contents

In [111]:
```
severe_boro_counts = boro_counts.sum(axis='columns')
severe_boro_counts
```

executed in 10ms, finished 15:02:20 2022-08-12

Out[111]:
```
BORO
Bronx            976
Brooklyn        2943
Manhattan       5747
Queens          2954
Staten Island    441
dtype: int64
```

In [112]:
```
boro_props = boro_counts.div(severe_boro_counts, axis='index')
boro_props
```

executed in 20ms, finished 15:02:20 2022-08-12

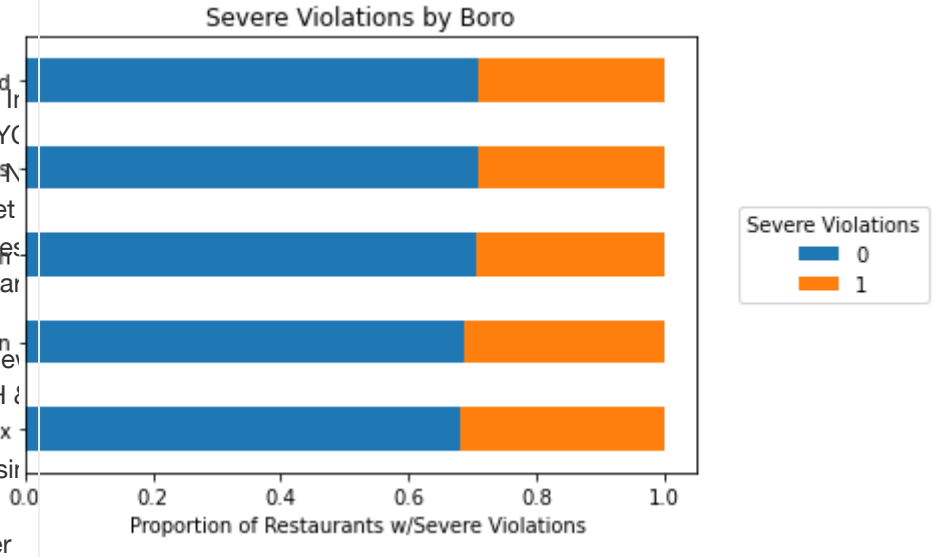
Out[112]:

Severe	0	1
BORO		
Bronx	0.68	0.32
Brooklyn	0.69	0.31
Manhattan	0.71	0.29
Queens	0.71	0.29
Staten Island	0.71	0.29

In [113]:

```
# Prototyping Stack Barh plot
ax = boro_props.plot.barh(stacked=True)
ax.set_title('Severe Violations by Boro')
ax.set_xlabel('Proportion of Restaurants w/Severe Violations ')
ax.legend(
    loc='center left',
    bbox_to_anchor=(1.05, 0.5),
    title='Severe Violations'
);
```

executed in 365ms, finished 15:02:20 2022-08-12



Contents

In [114]:

```
def violation_rate_plot(col, target, data, ax=None):
    """Stacked bar chart of severe health and safety violations rate ag
    each feature of the data.

    Args:
    col (string): column name of feature variable
    target (string): column name of target variable
    data (pandas DataFrame): dataframe that contains columns
        `col` and `target`
    ax (matplotlib axes object, optional): matplotlib axes
        object to attach plot to
    """
    counts = (eda_df[[target, col]]
              .groupby([target, col])
              .size()
              .unstack(target)
             )
    group_counts = counts.sum(axis='columns')
    props = counts.div(group_counts, axis='index')

    props.plot(kind="barh", stacked=True, ax=ax)

    ax.legend().remove()
```

executed in 11ms, finished 15:02:20 2022-08-12

In [115]:
```
1 eda_df.columns
```
executed in 11ms, finished 15:02:20 2022-08-12

Out[115]:
```
Index(['CAMIS', 'DBA', 'CUISINE DESCRIPTION', 'BORO', 'BUILDING', 'STREE
T',
       'PHONE', 'Latitude', 'Longitude', 'Community Board', 'Council Dist
rict',
       'Census Tract', 'Severe', 'id', 'review_count', 'rating', 'Review
s',
       'zipcode', 'prices', 'delivery', 'pickup', 'restaurant_reservatio
n'],
      dtype='object')
```

Contents

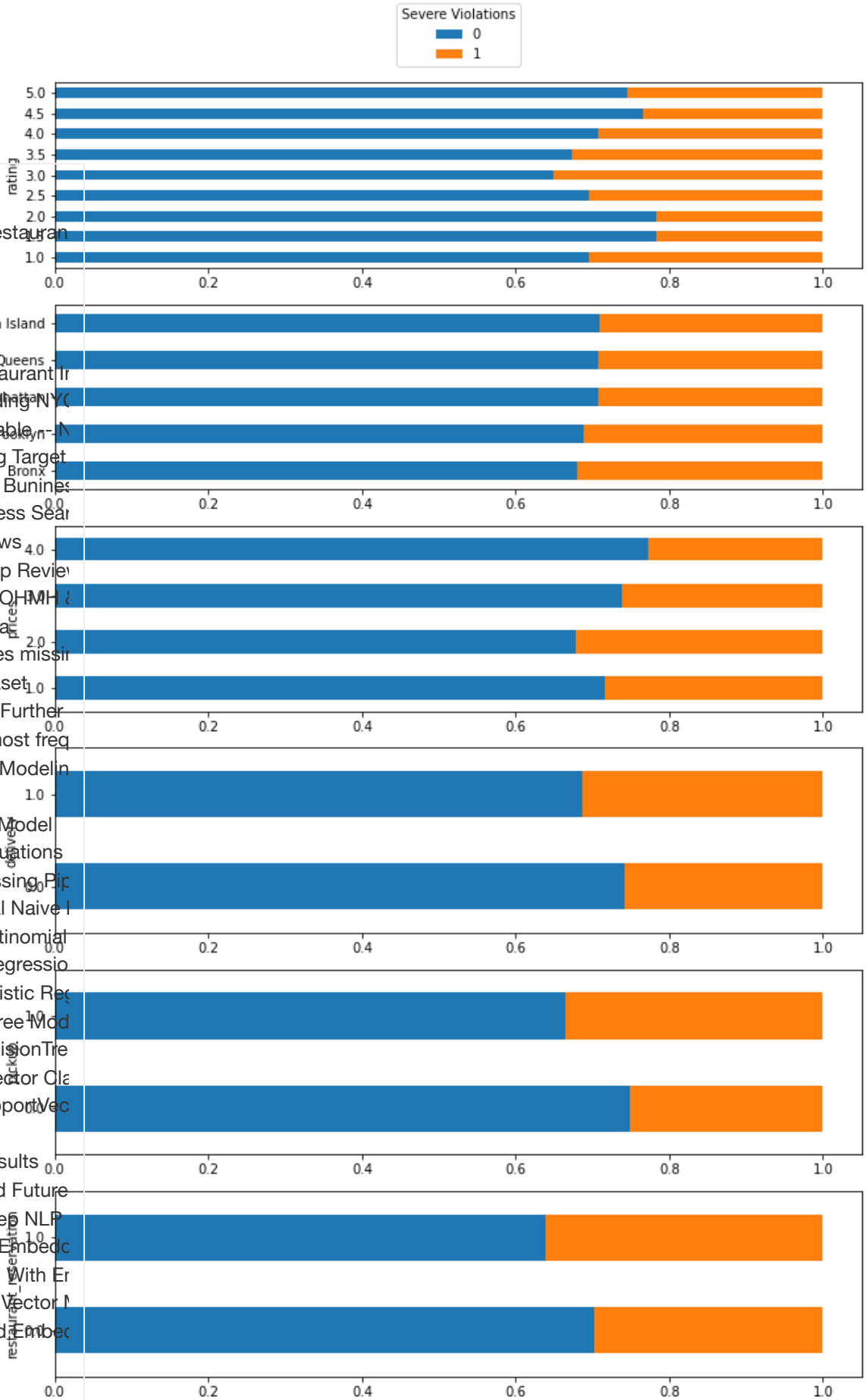
In [212]:

```
# Loop through several columns and plot against both severe violations.

cols_to_plot = [
    'rating',
    'BORO',
    'prices',
    'delivery',
    'pickup',
    'restaurant_reservation'
]

fig, ax = plt.subplots(
    len(cols_to_plot), figsize=(10,len(cols_to_plot)*2.5)
)
for idx, col in enumerate(cols_to_plot):

    violation_rate_plot(
        col, 'Severe', eda_df, ax=ax[idx]
    )


ax[0].legend(
    loc='lower center', bbox_to_anchor=(0.5, 1.05), title='Severe Viola
)
fig.tight_layout()
```

Executed in 1.44s, finished 19:05:19 2022-08-12

Contents

Contents

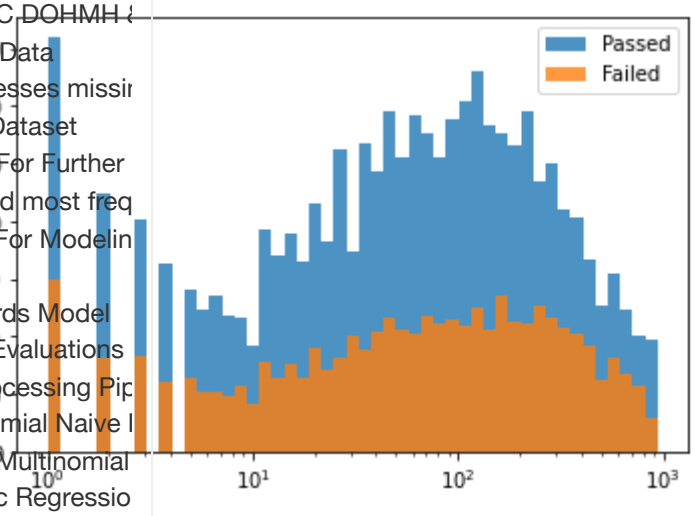


Contents

The plots above provide very little to no insight from plotting the target variable against the categorical variables.

In [116]:
```
plt.xscale('log')
bins =1.15**(np.arange(0,50))
plt.hist(eda_df[eda_df['Severe']==0]['review_count'],bins=bins,alpha=0.
plt.hist(eda_df[eda_df['Severe']==1]['review_count'],bins=bins,alpha=0.
plt.legend(('Passed','Failed'))
plt.show()
```

executed in 1.32s, finished 19:02:39 2022-08-12



This plot here attempts to show whether restaurants with more reviews are more likely to pass their inspection, however both target categories follow a similar curve.

In [117]:
```
# import street map
# https://data.cityofnewyork.us/City-Government/Borough-Boundaries/tqmj

street_map = gpd.read_file('data/Borough_Boundaries/geo_export_392103e7
```

executed in 526ms, finished 15:02:23 2022-08-12

In [118]:
```
geo_eda = eda_df.copy()
```

executed in 14ms, finished 15:02:23 2022-08-12

In [119]:
```
geo_eda.dropna(axis=0, subset=['Latitude','Longitude'], inplace=True)
```

executed in 21ms, finished 15:02:23 2022-08-12

Contents

In [120]:
```
# Remove the most extreme .1% latitudes, &
# the most extreme .1% longitudes

geo_eda = geo_eda[
 (geo_eda['Latitude'] >= np.percentile(geo_eda['Latitude'], 0.05)) &
 (geo_eda['Latitude'] < np.percentile(geo_eda['Latitude'], 99.95)) &
 (geo_eda['Longitude'] >= np.percentile(geo_eda['Longitude'], 0.05)) &
 (geo_eda['Longitude'] <= np.percentile(geo_eda['Longitude'], 99.95))
]
```
executed in 23ms, finished 15:02:23 2022-08-12

In [121]:
```
geo_eda = geo_eda[geo_eda['Longitude'] != 0]
```
executed in 16ms, finished 15:02:23 2022-08-12

In [122]:
```
geo_eda = geo_eda[geo_eda['Latitude'] != 0]
```
executed in 16ms, finished 15:02:23 2022-08-12

In [123]:
```
print(geo_eda['Longitude'].max())
print(geo_eda['Longitude'].min())
print(geo_eda['Latitude'].max())
print(geo_eda['Latitude'].min())
```
executed in 17ms, finished 15:02:23 2022-08-12

```
-73.70092857808
-74.24215580331187
40.91054991121208
40.511011068362
```

In [124]:
```
# designate coordinate system
crs = {'init':'epsg:4326'}
#zip x and y coordinates into single feature
geometry = [Point(xy) for xy in zip(geo_eda['Longitude'], geo_eda['Lati
```
executed in 243ms, finished 15:02:23 2022-08-12

In [125]:
```
# create GeoPandas dataframe
geo_df = gpd.GeoDataFrame(geo_eda, crs=crs, geometry = geometry)
```
executed in 658ms, finished 15:02:24 2022-08-12

In [126]:
```
# Plotting map
fig, ax = plt.subplots(figsize=(15,15))
street_map.plot(ax=ax, alpha=.5,color='gray')
geo_df[geo_df['Severe'] == 0].plot(ax=ax, alpha=.1, markersize=20,color
geo_df[geo_df['Severe'] == 1].plot(ax=ax, alpha=.1, markersize=20,color
plt.legend(prop={'size':15});
```

executed in 5.42s, finished 15:02:29 2022-08-12

Contents

We already learned from the plot above there is very little we can infer from the restaurant location by each borough. And the map plot above confirms there is no insight to be gained from location data in all. There are clearly a much higher density of restaurants in Manhattan, especially Midtown and Lower Manhattan as well as the neighborhoods along the East River in Brooklyn and Queens. But the data points don't tell us were are higher portion of restaurants fail their health inspections.

6 Preprocessing For Further EDA

6.0.1 Lets find most frequent words in Negative reviews

Basically, we will find most frequent words in reviews to get an overview of why users gave low ratings. These words could be related to those business attributes or services about which users are most unhappy.

In [127]:
```
txt_eda = df_1.copy()
```
executed in 17ms, finished 15:02:29 2022-08-12

In [128]:
```
passed_df = txt_eda.loc[txt_eda['Severe']==0]
passed_df
```
executed in 130ms, finished 15:02:29 2022-08-12

13058	50113198	SENG SEAFOOD RESTAURANT	Seafood	Manhattan	39	EAST BROADWAY	+164679
13059	50117350	SONG TEA	Coffee/Tea	Manhattan	488	7 AVENUE	+164666

In [129]:
```
failed_df = txt_eda.loc[txt_eda['Severe']==1]
failed_df
```
executed in 92ms, finished 15:02:29 2022-08-12

Out[129]:

	CAMIS	DBA	CUISINE DESCRIPTION	BORO	BUILDING	STREET	PHONE	La
5	40360076	CARVEL	Frozen Desserts	Brooklyn	203	CHURCH AVENUE	+17184389501	

Contents

In [130]:
```
passed_corpus = passed_df['Reviews'].to_list()
passed_corpus[:5]
```
executed in 29ms, finished 15:02:29 2022-08-12

```
['remony progression, reception outline, and meal-music timing, the coordin
ation was flawless. I, as the bride, didn't have to worry about a thing o
n my wedding day. Everything was perfect because of the Riviera team. Man
y of our guests stated that it was the best wedding they had every attend
ed :) Couldn't have wished for a more memorable night. We would recommend
the Riviera team in a heartbeat!.Tommy is awesome! My fiancé and I met wi
th him in hopes of having our wedding here. Out of the 15 venues we visit
ed, Tommy was the most down to earth and relatable person we met with. Th
e venue is gorgeous and the menu is jam packed with loads of options. The
backyard is so stunning and will make for great photo backgrounds. We end
ed up going with a venue outside of Brooklyn but we will definitely keep
in contact in hopes to use this beautiful venue at a later date..Magica
l!!!!!!!!! No need to say more!!!My cousin got married on 6/23/18Wow Rivi
era! You really did an amazing job!From the food to the smallest details!
Like a bottle of champagne in the bathrooms so the guest dont go thirsty.
lol The food outstanding! I had the rack of lamb! This was the best lamb
I ever had! And the potato purée?!!!!!! Mouth watering! Great job Rivier
a! The staff was outstanding!!!!! The drinks were served all night! And t
he steps kept on coming to our table without asking! Which was a great!!
```

In [131]:
```
failed_corpus = failed_df['Reviews'].to_list()
failed_corpus[:5]
```

executed in 23ms, finished 15:02:29 2022-08-12

Out[131]:
```
["I love carvel there used to be carvel at Nostrand Avenue and Church Ave
nue back in the nineties that one closed down, but this location is locat
ed in the Kensington section of Brooklyn .there ice cream is great, I lov
e coming here to this location I'll definitely mind coming back in heartb
eat. Staff members are cool too! \xa0(their delivery here as well I'm not
sure if it's from Grubhub, uber eats, door dash, postmates, etc )- call i
n double check P.S. something about Carvel ice cream that does it, no off
ense Baskin Robbins .....Lol !!!!( all dough I could go to the shop and s
top supermarket and get a carvel ice cream cake box I prefer to come her
e, instead ).Very convenient by the MTA transportation:Subway:F, G-Train
s. Church avenue. Train station.Bus:B35 ,B67,B103.Had the vanilla ice cre
am it tasted off ... Now I'm home at 3am in the morning with food poisoni
ng. Be wary..Cousin & I went to polish off our dinner with some ice crea
m. Best sundaes we ever had. The amount of fudge was nothing I'd ever see
n. The lady who made it put my peanuts on first, then the hot fudge. At f
irst I was confused but it made sense. Instead of the little suckers esca
ping and falling off, they were mounted underneath the mountain of fudge,
guaranteed to be eaten with no escapees..Since the ice cream is/should be
the same in all locations the 3 is a review of the store. The store is a
```

Contents

In [132]:
```
# Tokenizing the corpus of review text from restaurants that have sever
failed_tokens = word_tokenize(','.join(failed_corpus))
```

executed in 52.5s, finished 15:03:22 2022-08-12

In [133]:
```
# Tokenizing the corpus of review text from restaurants that have not s
passed_tokens = word_tokenize(','.join(passed_corpus))
```

executed in 1m 38.1s, finished 15:05:20 2022-08-12

In [134]:
```
# Importing English stop words
stopwords_list = stopwords.words('english')
# Add punctuation marks to the stopwords_list
stopwords_list.extend(string.punctuation)
additional_punc = ['“','”','...',"''",'’','``']
stopwords_list.extend(additional_punc)
restaurant_words = ['restaurant', 'place', 'food', 'drink']
stopwords_list.extend(restaurant_words)
```

executed in 33ms, finished 15:05:20 2022-08-12

In [135]:
```
# Removing stopwords from failed_corpus
stopped_failed_tokens = [w.lower() for w in failed_tokens if w.lower()
stopped_failed_tokens[:10]
```

executed in 13.3s, finished 15:05:33 2022-08-12

Out[135]:
```
['love',
 'carvel',
 'used',
 'carvel',
 'nostrand',
 'avenue',
 'church',
 'avenue',
 'back',
 'nineties']
```

Contents

In [136]:
```
# Removing stopwords from passed_corpus
stopped_passed_tokens = [w.lower() for w in passed_tokens if w.lower()
stopped_passed_tokens[:10]
```

executed in 29.8s, finished 15:06:03 2022-08-12

Out[136]:
```
['forgot',
 'mom',
 'beer',
 'burger',
 'tasted',
 'weird',
 'give',
 'stars',
 'absolutely',
 'would']
```

In [137]:
```
# Creating FreqDist from stopped_failed_tokens
freq_failed = FreqDist(stopped_failed_tokens)
freq_failed.most_common(15)
```

executed in 2.06s, finished 15:06:05 2022-08-12

Out[137]:
```
[('food', 27371),
 ('like', 24690),
 ('one', 22707),
 ('good', 20802),
 ('like', 13681),
 ('great', 12793),
 ('one', 11126),
 ('order', 11084),
 ('chicken', 10993),
 ('would', 10692),
 ('time', 10682),
 ('get', 10465),
 ('service', 10413),
 ('would', 10159),
 ('really', 9961)]
```

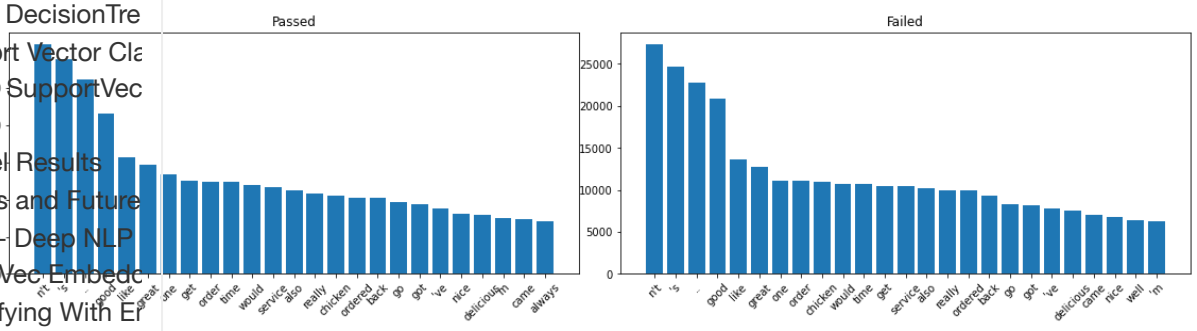
In [138]:
```
# Creating FreqDist from stopped_passed_tokens
freq_passed = FreqDist(stopped_passed_tokens)
passed_bar = freq_passed.most_common(15)
```
executed in 4.40s, finished 15:06:10 2022-08-12

In [139]:
```
# Creating FreqDist from stopped_passed_tokens
passed_bar_count = [x[1] for x in freq_passed.most_common(25)]
passed_bar_words = [x[0] for x in freq_passed.most_common(25)]
```
executed in 663ms, finished 15:06:10 2022-08-12

In [140]:
```
# Creating FreqDist from stopped_failed_tokens
failed_bar_count = [x[1] for x in freq_failed.most_common(25)]
failed_bar_words = [x[0] for x in freq_failed.most_common(25)]
```
executed in 55ms, finished 15:06:10 2022-08-12

In [ ]:
```

```
executed in 13ms, finished 22:31:05 2022-08-10

In [141]:
```
new_fig = plt.figure(figsize=(16,4))

ax = new_fig.add_subplot(121)
ax2 = new_fig.add_subplot(122)

ax.bar(passed_bar_words,passed_bar_count)
ax2.bar(failed_bar_words,failed_bar_count)

ax.title.set_text('Passed')
ax2.title.set_text('Failed')

for ax in new_fig.axes:
    plt.sca(ax)
    plt.xticks(rotation=45)

plt.tight_layout(pad=0)

plt.show()
```
executed in 1.33s, finished 15:06:12 2022-08-12



Contents

In [142]:

```
# Functionizing wordcloud generator
def wordcloud_generator(tokens, collocations=False, background_color='b
                        colormap='Reds', display=True):


    # Initalize a WordCloud
    wordcloud = WordCloud(collocations=collocations,
                          background_color=background_color,
                          colormap=colormap,
                          width=500, height=300)

    # Generate wordcloud from tokens
    wordcloud.generate(','.join(tokens))

    # Plot with matplotlib
    if display:
        plt.figure(figsize = (12, 15), facecolor = None)
        plt.imshow(wordcloud)
        plt.axis('off');

    return wordcloud
```

executed in 12ms, finished 15:06:12 2022-08-12

In [143]:

```
# Generating a WordCloud for reviews from restaurants that have failed
failed_cloud = wordcloud_generator(stopped_failed_tokens, collocations=
```

executed in 18.6s, finished 15:06:30 2022-08-12

In [144]:
```
# Generate a WordCloud from reviews of restaurants that have not severe
passsed_cloud = wordcloud_generator(stopped_passed_tokens, colormap='Gr
                                    collocations=True)
```

executed in 43.0s, finished 15:07:13 2022-08-12

7 Preprocessing For Modeling

In [384]:
```
df_1.columns
```

executed in 33ms, finished 21:02:57 2022-08-15

Out[384]:
```
Index(['CAMIS', 'DBA', 'CUISINE DESCRIPTION', 'BORO', 'BUILDING', 'STREE
       'PHONE', 'Latitude', 'Longitude', 'Community Board', 'Council Dist
       'Census Tract', 'Severe', 'id', 'alias', 'name', 'image_url',
       'is_closed', 'url', 'review_count', 'categories', 'rating',
       'coordinates', 'transactions', 'price', 'location', 'phone',
       'display_phone', 'Reviews', 'zipcode'],
      dtype='object')
```

In [385]:
```
# Dropping unneccessary columns
model_df = df_1.drop(columns=['CAMIS', 'DBA', 'CUISINE DESCRIPTION', 'B
       'PHONE', 'Latitude', 'Longitude', 'Community Board', 'Council Di
       'Census Tract', 'alias', 'name', 'image_url',
       'is_closed', 'url', 'review_count', 'categories', 'rating',
       'coordinates', 'transactions', 'price', 'location', 'phone',
       'display_phone', 'zipcode'])
```

executed in 135ms, finished 21:02:58 2022-08-15

In [386]:
```
model_df.head(3)
```

executed in 27ms, finished 21:02:58 2022-08-15

Out[386]:

	Severe	id	Reviews
0	0	AMxMPBkWi20dn_1BRalahA	Forgot my root beer and the burger tasted very weird.If I can give this place no stars I absolutely would. The workers here are in efficient. They do not work the best possible way. They cause th...
1	0	g8fAhVqBsC-BN6otcn85OA	I had my wedding here a month ago and I would give Riviera 10 stars if I could. It was such a joy and ease to work with Adam, Tommy, and their staff. A true diamond in the rough of Coney Island a...
2	0	Pe6MsH2DW0CXjvUtxUpl4A	We've recently discovered this gem on Uber Eats and they have the best sandwiches!!The broccoli rabe fried eggplant mozzarella sandwich was amazing. Having a fresh green veggie paired with a light...

In [387]:
```
model_df['Reviews'].isna().sum()
```

executed in 10ms, finished 21:02:59 2022-08-15

Out[387]: 0

In [388]:
```
# Checking Class Balance
model_df['Severe'].value_counts()
```

executed in 14ms, finished 21:03:00 2022-08-15

Out[388]:
```
0    91158
1    39103
Name: Severe, dtype: int64
```

In [389]:
```
# Make X and y
y = model_df['Severe'].copy()
X = model_df['Reviews'].copy()
```

executed in 11ms, finished 21:03:02 2022-08-15

In [396]:
```
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
                                                    random_state=42)
```

executed in 36ms, finished 21:08:59 2022-08-15

8 Models

8.1 Bag-of-Words Model

8.1.1 Model Evaluations

Because the objective of this project is to flag restaurants that are likely to fail their health inspections the metric we will be optimizing for is Recall because we need our model to find all of the restaurants that pose a risk to public health. This metric is defined by the number of true positives divided by true positives and false negatives. If our model mistakenly classifies a clean kitchen as unsafe, the consequence of sending an inspector to check on it is much less worse than the alternative of classifying an unsafe establishment as clean. The Accuracy measurement, scoring how many times the model correctly predicted is a good indicator of performance but not as suitable as recall because of this business case AND because of our class imbalance. That is to say, since only 30% of our businesses are flagged. If our model predicts every restaurant will pass its inspection, it will be correct 70%.

Contents

In [381]:

```python
def evaluate_classification(model, X_test,y_test,cmap='Blues',
                            normalize=None,classes=['Passed','Failed'],
                            X_train = None, y_train = None,):
    """Evaluates a scikit-learn binary classification model.

    Args:
        model (classifier): any sklearn classification model.
        X_test (Frame or Array): X data
        y_test (Series or Array): y data
        cmap (str, optional): Colormap for confusion matrix. Defaults t
        normalize (str, optional): normalize argument for plot_confusio
                                   Defaults to 'true'.
        classes (list, optional): List of class names for display. Defa
        figsize (tuple, optional): figure size Defaults to (8,4).

        X_train (Frame or Array, optional): If provided, compare model.
                                  for train and test. Defaults to None.
        y_train (Series or Array, optional): If provided, compare model
                                  for train and test. Defaults to None.


    ## Get Predictions and Classification Report
    y_hat_test = model.predict(X_test)
    print(metrics.classification_report(y_test, y_hat_test,target_names

    ## Plot Confusion Matrid and roc curve
    fig,ax = plt.subplots(ncols=2, figsize=figsize)
    metrics.plot_confusion_matrix(model, X_test,y_test,cmap=cmap,
                                  normalize=normalize,display_labels=cl
                                  ax=ax[0])

    ## if roc curve erorrs, delete second ax
    try:
        curve = metrics.plot_roc_curve(model,X_test,y_test,ax=ax[1])
        curve.ax_.grid()
        curve.ax_.plot([0,1],[0,1],ls=':')
        fig.tight_layout()
    except:
        fig.delaxes(ax[1])
    plt.show()

    ## Add comparing Scores if X_train and y_train provided.
    if (X_train is not None) & (y_train is not None):
        print(f"Training Score = {model.score(X_train,y_train):.2f}")
        print(f"Test Score = {model.score(X_test,y_test):.2f}")
```

executed in 15ms, finished 21:01:48 2022-08-15

Contents

In [382]:
```
def plot_importance(tree, X_train, top_n=20,figsize=(10,10)):

    df_importance = pd.Series(tree.feature_importances_,
                              index=X_train.columns)
    df_importance.sort_values(ascending=True).tail(top_n).plot(
        kind='barh',figsize=figsize,title='Feature Importances',
    ylabel='Feature',)
    return df_importance
```

executed in 51ms, finished 21:02:05 2022-08-15

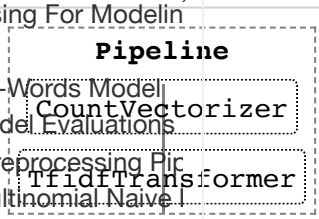
8.2 Text Preprocessing Pipeline

Here we'll make a text preprocessing pipeline using scikit-learns term frequency inverse document
to give more important words more weight.

In [436]:
```
text_pipe = Pipeline(steps=[
    ('count_vectorizer',CountVectorizer(min_df=.01,max_df=.99,
                                        token_pattern=r"([a-zA-Z]+(?:'[
                                        lowercase=True,
                                        stop_words=stopwords_list,
                                        ngram_range=(1,2))),
    ('tf_transformer',TfidfTransformer(use_idf=True))
])
text_pipe
```

executed in 44ms, finished 22:09:38 2022-08-15

Out[436]:



In [437]:
```
## Test out the text pipeline on X_train
X_train_pipe = text_pipe.fit_transform(X_train)
X_test_pipe = text_pipe.transform(X_test)
X_train_pipe
```

executed in 29.6s, finished 22:10:08 2022-08-15

Out[437]:
```
<9142x6236 sparse matrix of type '<class 'numpy.float64'>'
        with 2998698 stored elements in Compressed Sparse Row format>
```

In [440]:
```
X_train_pipe.get_shape
```

executed in 19ms, finished 22:11:17 2022-08-15

Out[440]:
```
<bound method spmatrix.get_shape of <9142x6236 sparse matrix of type '<cl
ass 'numpy.float64'>'
        with 2998698 stored elements in Compressed Sparse Row format>>
```

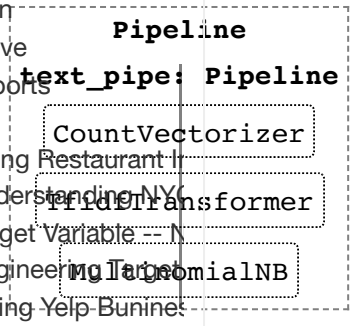
8.2.1 Multinomial Naive Bayes Model

We'll being the text classification problem with a Naive Bayes Model, which typically does well with text classifications.

In [293]:
```
## Make a full pipeline with the DecisionTree model as the second step
mnb_pipe = Pipeline([('text_pipe',text_pipe),
                     ('clf',MultinomialNB())])
mnb_pipe
```

executed in 75ms, finished 18:26:36 2022-08-15

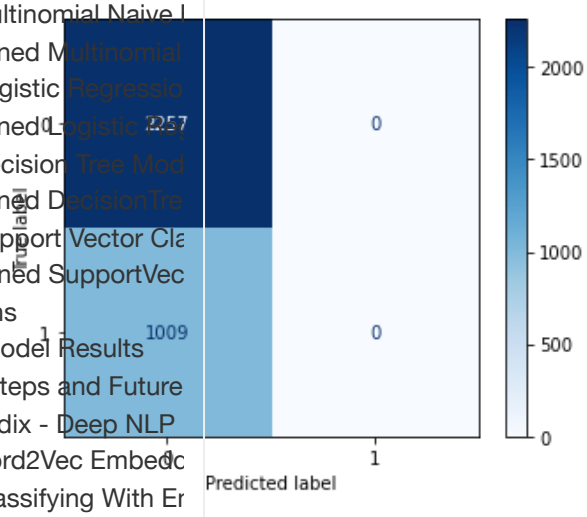
Out[293]:

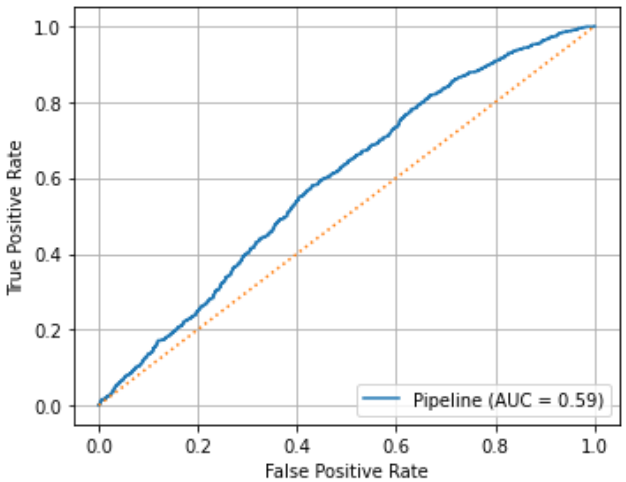


In [294]:
```
##Modeling with full pipeline
mnb_pipe.fit(X_train,y_train)
evaluate_classification(mnb_pipe,X_test,y_test,X_train=X_train, y_train
```

executed in 57.7s, finished 18:27:34 2022-08-15

	precision	recall	f1-score	support
0	0.69	1.00	0.82	2257
1	0.00	0.00	0.00	1009
accuracy			0.69	3266
macro avg	0.35	0.50	0.41	3266
weighted avg	0.48	0.69	0.56	3266





Training Score = 0.70
Test Score = 0.69

The MultinomialNB model gas a 70% accuracy, but did not predict any restaurants failed their inspection. This is not helpful for the objective of this project. We'll try tuning the hyperparameters with Gridsearch.

Tuning with GridsearchCV

In [258]:
```
params = {'clf__alpha': [0.001, 0.01, 0.1, 1]}

gs = GridSearchCV(mnb_pipe, param_grid=params,
                                  scoring='recall')

gs.fit(X_train,  y_train)
gs.best_params_
```

executed in 3m 46s, finished 17:23:25 2022-08-15

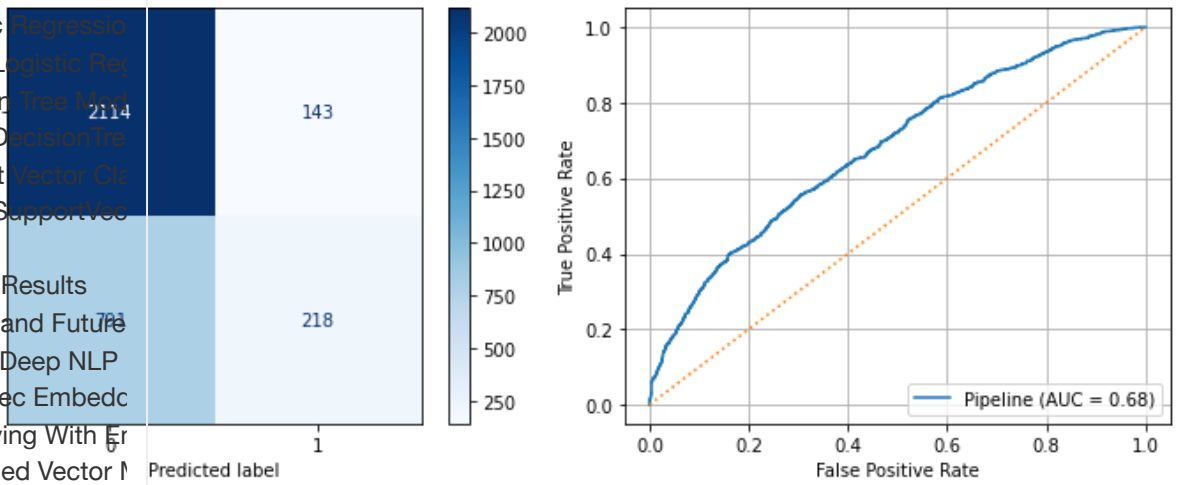
Out[258]:
```
{'clf__alpha': 0.001}
```

8.2.2 Tuned Multinomial Naive Bayes Model

In [259]:
```
tuned_mnb_pipe = Pipeline([('text_pipe',text_pipe),
                           ('clf',MultinomialNB(alpha=0.001))])

tuned_mnb_pipe.fit(X_train,y_train)
evaluate_classification(tuned_mnb_pipe,X_test,y_test,X_train=X_train,
                        y_train=y_train)
```

executed in 37.0s, finished 17:25:12 2022-08-15

```
              precision    recall  f1-score   support

           0       0.73      0.94      0.82      2257
           1       0.60      0.22      0.32      1009

    accuracy                           0.71      3266
   macro avg       0.67      0.58      0.57      3266
weighted avg       0.69      0.71      0.66      3266
```




```
Training Score = 0.96
Test Score = 0.71
```

Tuning the alpha smoothing parameter did improve the models accuracy from 70% to 96% in the training data, but only slightly improved the score for the validation data. This model it overfit to the

training data and will perform poorly at classifying new reviews it hasn't seen. It did however predict some violations but the recall score of 22% is below satisfactory.

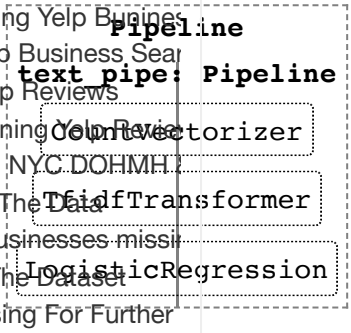
8.2.3 Logistic Regression Model

The next classifier we'll fit to the data is a Logistic Regression model. A logistic regression can provide a useful tool to our stakeholders by not just predicting whether or not a restaurant will fail its health and safety inspection but can model the probability of a restaurant failing its inspection.

In [442]:
```
## Make a full pipeline with the LogisticRegression model as the second
lr_pipe = Pipeline([('text_pipe',text_pipe),
                    ('clf',LogisticRegression(solver='sag',class_weigh
lr_pipe
```

executed in 217ms, finished 22:12:22 2022-08-15

Out[442]:

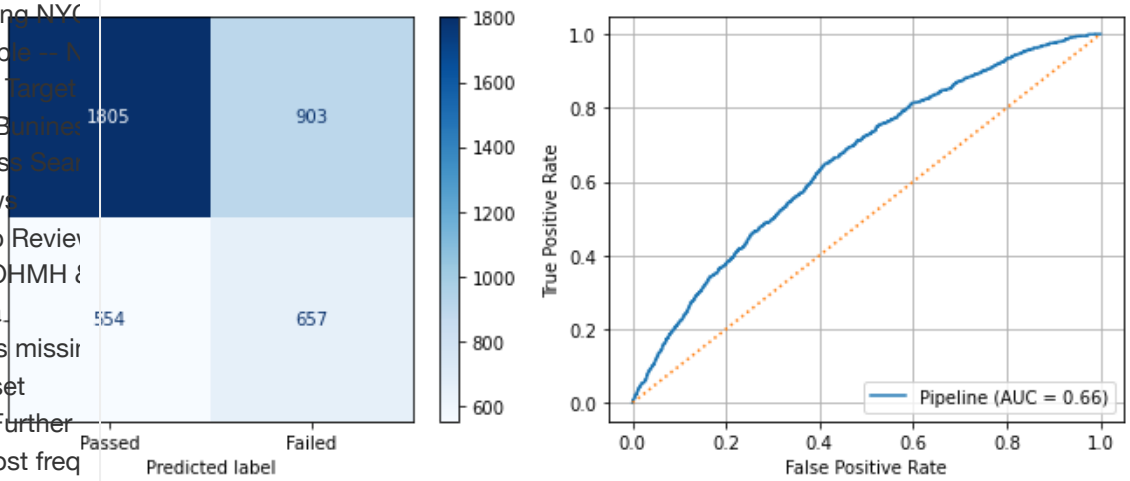


Contents

In [443]:
```
## Modeling with full pipeline
lr_pipe.fit(X_train,y_train)
evaluate_classification(lr_pipe,X_test,y_test,X_train=X_train, y_train=
```

executed in 1m 1.91s, finished 22:13:26 2022-08-15

```
              precision    recall  f1-score   support

      Passed       0.77      0.67      0.71      2708
      Failed       0.42      0.54      0.47      1211

    accuracy                           0.63      3919
   macro avg       0.59      0.60      0.59      3919
weighted avg       0.66      0.63      0.64      3919
```




```
Training Score = 0.78
Test Score = 0.63
```

The baseline Logistic Regression shows more promising results than the tuned Naive Bayes model. Its definitely overfitting. Lets try to get more performance from the model by tuning the hyperparameters.

Hyperparameter Tuning With GridsearchCV

Contents

In [447]:
```
1 lr_pipe.get_params().keys()
```
executed in 64ms, finished 22:45:32 2022-08-15

Out[447]:
```
dict_keys(['memory', 'steps', 'verbose', 'text_pipe', 'clf', 'text_pipe__
memory', 'text_pipe__steps', 'text_pipe__verbose', 'text_pipe__count_vect
orizer', 'text_pipe__tf_transformer', 'text_pipe__count_vectorizer__analy
zer', 'text_pipe__count_vectorizer__binary', 'text_pipe__count_vectorizer
__decode_error', 'text_pipe__count_vectorizer__dtype', 'text_pipe__count_
vectorizer__encoding', 'text_pipe__count_vectorizer__input', 'text_pipe__
count_vectorizer__lowercase', 'text_pipe__count_vectorizer__max_df', 'tex
t_pipe__count_vectorizer__max_features', 'text_pipe__count_vectorizer__mi
n_df', 'text_pipe__count_vectorizer__ngram_range', 'text_pipe__count_vect
orizer__preprocessor', 'text_pipe__count_vectorizer__stop_words', 'text_p
ipe__count_vectorizer__strip_accents', 'text_pipe__count_vectorizer__toke
n_pattern', 'text_pipe__count_vectorizer__tokenizer', 'text_pipe__count_v
ectorizer__vocabulary', 'text_pipe__tf_transformer__norm', 'text_pipe__tf
_transformer__smooth_idf', 'text_pipe__tf_transformer__sublinear_tf', 'te
xt_pipe__tf_transformer__use_idf', 'clf__C', 'clf__class_weight', 'clf__d
ual', 'clf__fit_intercept', 'clf__intercept_scaling', 'clf__l1_ratio', 'c
lf__max_iter', 'clf__multi_class', 'clf__n_jobs', 'clf__penalty', 'clf__r
andom_state', 'clf__solver', 'clf__tol', 'clf__verbose', 'clf__warm_star
t'])
```

Contents

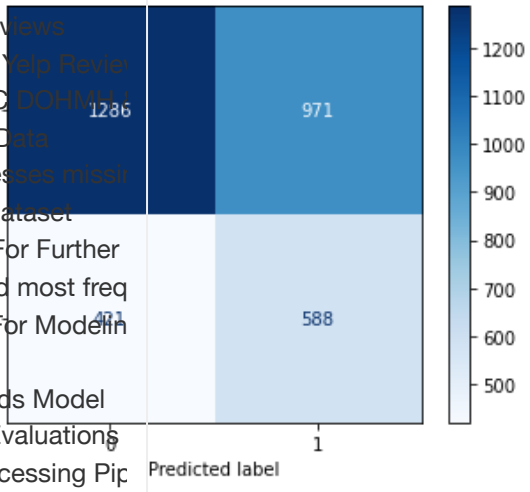
8.2.4 Tuned Logistic Regression Model

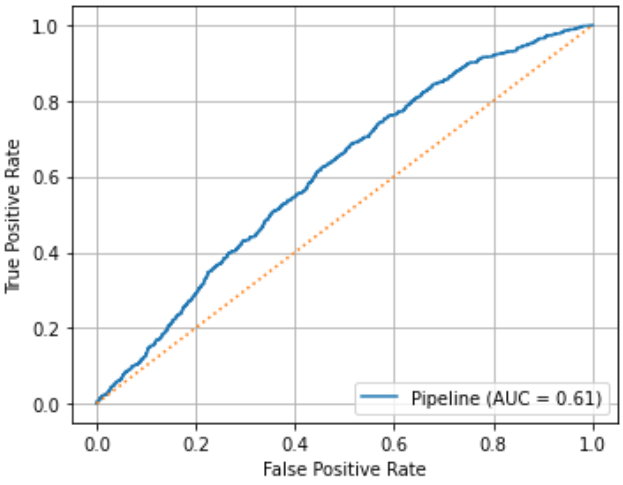
In [301]:
```
tuned_lr_pipe = Pipeline([('text_pipe',text_pipe),
                          ('clf',LogisticRegression(C=0.1,max_iter=100
                                                    solver='saga',
                                                    class_weight='bala


tuned_lr_pipe.fit(X_train,y_train)
evaluate_classification(tuned_lr_pipe,X_test,y_test,X_train=X_train,
                        y_train=y_train)
```

executed in 1m 15.2s, finished 19:06:08 2022-08-15

Contents

	precision	recall	f1-score	support
0	0.75	0.57	0.65	2257
1	0.38	0.58	0.46	1009
accuracy			0.57	3266
macro avg	0.57	0.58	0.55	3266
weighted avg	0.64	0.57	0.59	3266






```
Training Score = 0.63
Test Score = 0.57
```

Contents

In [446]:

```
params = {'text_pipe__tf_transformer__use_idf':[True, False],
          'text_pipe__count_vectorizer__token_pattern':[None,r"([a-zA-Z
          'text_pipe__count_vectorizer__stop_words':[None,stopwords_lis
          'clf__class_weight': ['balanced'],
          'clf__max_iter': [100, 500],
          'clf__C': [0.1, 1],
          'clf__solver': ['lbfgs', 'sag', 'saga']}

gs = GridSearchCV(estimator=lr_pipe, param_grid = params,
                  scoring='recall',verbose=3,n_jobs=-1,cv=3)

gs.fit(X_train,  y_train)
gs.best_params_
```

executed in 30m 8s, finished 22:45:32 2022-08-15

```
Fitting 3 folds for each of 96 candidates, totalling 288 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent worker
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed:  2.7min
[Parallel(n_jobs=-1)]: Done 120 tasks      | elapsed: 12.5min
[Parallel(n_jobs=-1)]: Done 280 tasks      | elapsed: 28.1min
[Parallel(n_jobs=-1)]: Done 288 out of 288 | elapsed: 29.5min finished
```

Out[446]:

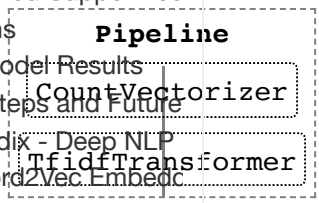
```
{'clf__C': 0.1,
 'clf__class_weight': 'balanced',
 'clf__max_iter': 100,
 'clf__solver': 'lbfgs',
 'text_pipe__count_vectorizer__stop_words': None,
 'text_pipe__count_vectorizer__token_pattern': "([a-zA-Z]+(?:'[a-z]+)?)",
 'text_pipe__tf_transformer__use_idf': False}
```

In [448]:

```
text_pipe2 = Pipeline(steps=[
        ('count_vectorizer',CountVectorizer(min_df=.01,max_df=.99,
                                            token_pattern=r"([a-zA-Z]+(?:'[
                                            lowercase=True,
                                            stop_words=None,
                                            ngram_range=(1,2))),
        ('tf_transformer',TfidfTransformer(use_idf=False))
    ])
text_pipe2
```

executed in 98ms, finished 22:50:27 2022-08-15

Out[448]:

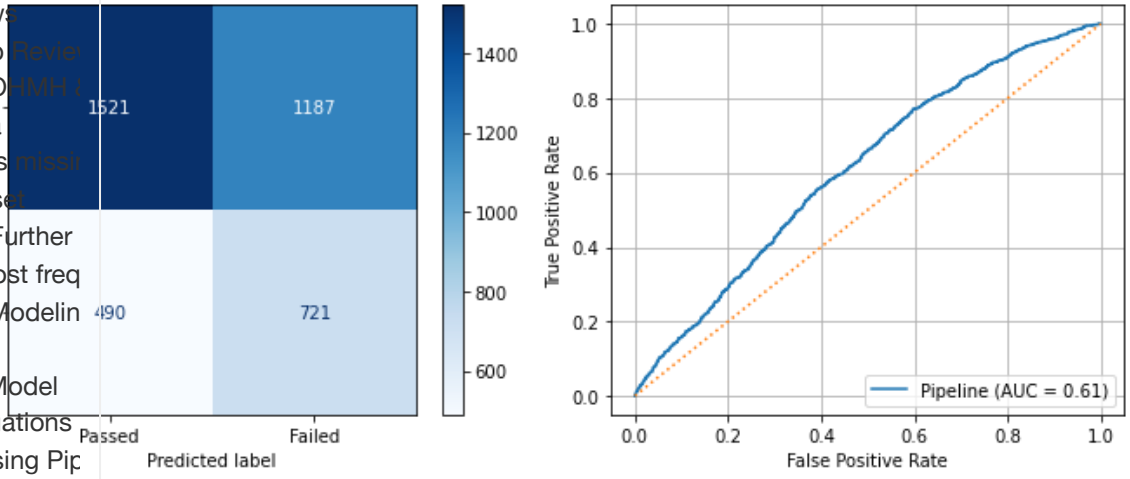


In [449]:
```
tuned_lr_pipe2 = Pipeline([('text_pipe2',text_pipe2),
                           ('clf',LogisticRegression(C=0.1,max_iter=100
                                                     solver='lbfgs',
                                                     class_weight='bala


tuned_lr_pipe2.fit(X_train,y_train)
evaluate_classification(tuned_lr_pipe2,X_test,y_test,X_train=X_train,
                        y_train=y_train)
```

executed in 1m 29.1s, finished 22:52:11 2022-08-15

Contents


```
              precision    recall  f1-score   support

      Passed       0.76      0.56      0.64      2708
      Failed       0.38      0.60      0.46      1211

    accuracy                           0.57      3919
   macro avg       0.57      0.58      0.55      3919
weighted avg       0.64      0.57      0.59      3919
```




```
Training Score = 0.59
Test Score = 0.57
```

By tuning the Logistic Regression model we were able to increase Recall by 6% but this cost our accuracy to drop 6% on the validation data. Which is a good trade-off for the business problem.

In [450]:
```
y_actual = y_test.reset_index(drop=True)
```

executed in 14ms, finished 22:54:42 2022-08-15

In [453]:
```
lr_probas = tuned_lr_pipe2.predict_proba(X_test)
```
executed in 8.41s, finished 22:54:52 2022-08-15

In [454]:
```
probs_df = pd.DataFrame(lr_probas, columns= ['Pass', 'Fail'])
probs_df = pd.concat([probs_df,y_actual],axis=1)
probs_df
```
executed in 26ms, finished 22:54:52 2022-08-15

Out[454]:

	Pass	Fail	Severe
0	0.60	0.40	0
1	0.48	0.52	0
2	0.57	0.43	0
3	0.43	0.57	0
4	0.45	0.55	0
...	...	...	...
3914	0.43	0.57	1
3915	0.46	0.54	1
3916	0.49	0.51	0
3917	0.47	0.53	0
3918	0.45	0.55	0

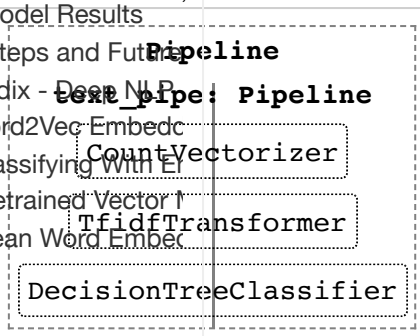
3919 rows × 3 columns

Here above the model returns the probability a restaurant will pass or fail its inspection. This can be useful to inspectors who will be visiting all businesses but can prioritize those most at risk.

8.2.5 Decision Tree Model

In [330]:
```
## Make a full pipeline with the DecisionTree model as the second step
dt_pipe = Pipeline([('text_pipe',text_pipe),
                    ('clf',DecisionTreeClassifier(max_depth=6,class_we
dt_pipe
```
executed in 86ms, finished 19:25:24 2022-08-15

Out[330]:

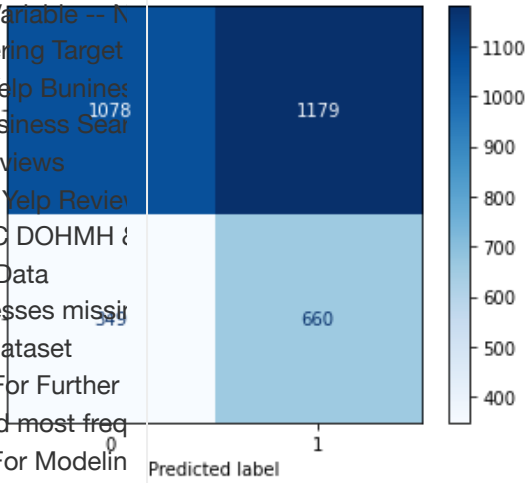


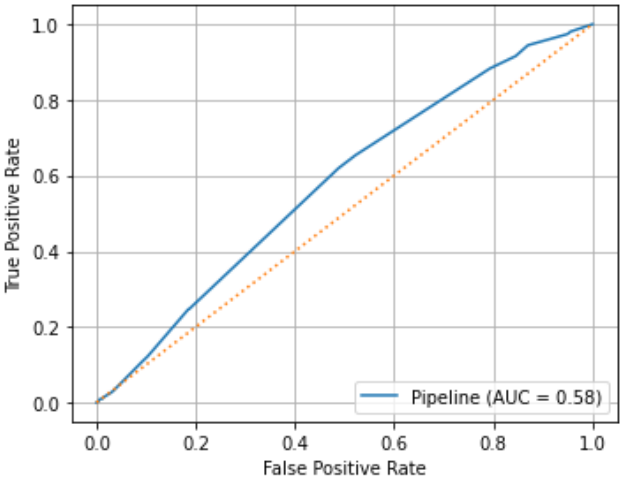
Contents

In [333]:
```
## Modeling with full pipeline
dt_pipe.fit(X_train,y_train)
evaluate_classification(dt_pipe,X_test,y_test,X_train=X_train, y_train=
```

executed in 55.8s, finished 19:30:48 2022-08-15

	precision	recall	f1-score	support
0	0.76	0.48	0.59	2257
1	0.36	0.65	0.46	1009
accuracy			0.53	3266
macro avg	0.56	0.57	0.52	3266
weighted avg	0.63	0.53	0.55	3266





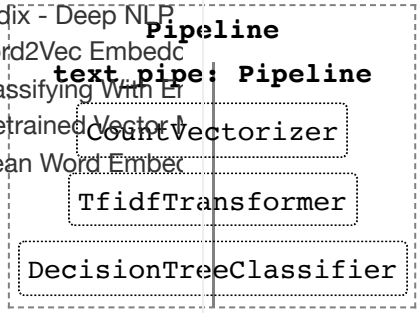

```
Training Score = 0.57
Test Score = 0.53
```

In [ ]:
```

```

In [466]:
```
## Make a full pipeline with the DecisionTree model as the second step
dt_pipe = Pipeline([('text_pipe',text_pipe),
                    ('clf',DecisionTreeClassifier(max_depth=6,class_we
dt_pipe
```

executed in 478ms, finished 01:25:20 2022-08-16

Out[466]:

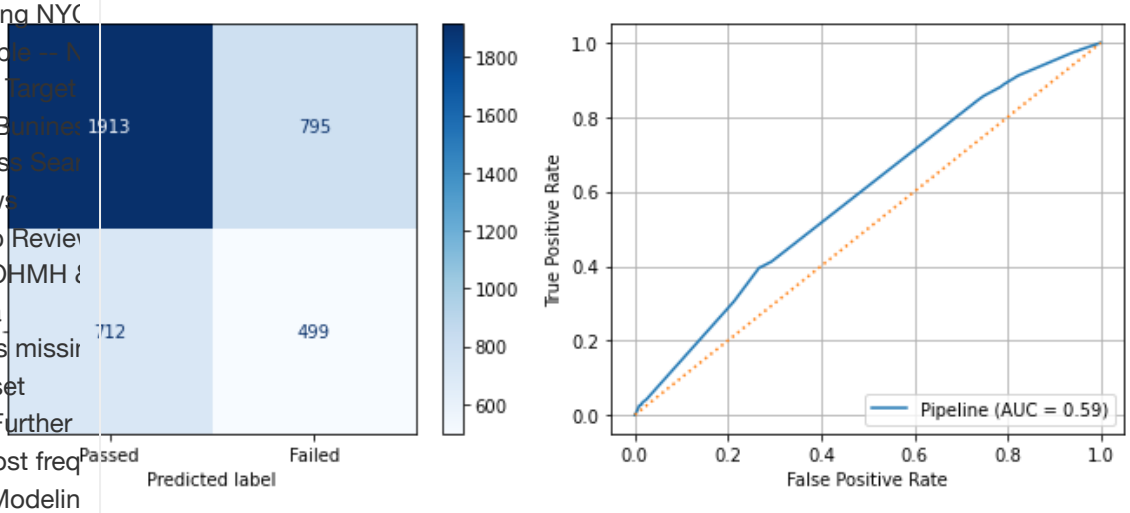


Contents

In [467]:
```
## Modeling with full pipeline
dt_pipe.fit(X_train,y_train)
evaluate_classification(dt_pipe,X_test,y_test,X_train=X_train, y_train=
```

executed in 1m 21.3s, finished 01:26:43 2022-08-16

```
              precision    recall  f1-score   support

      Passed       0.73      0.71      0.72      2708
      Failed       0.39      0.41      0.40      1211

    accuracy                           0.62      3919
   macro avg       0.56      0.56      0.56      3919
weighted avg       0.62      0.62      0.62      3919
```

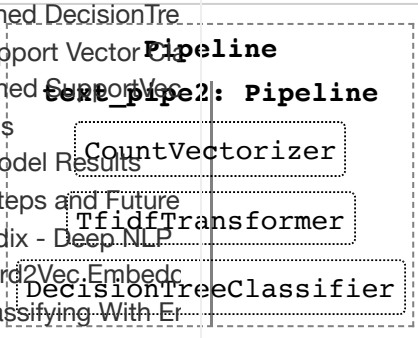



```
Training Score = 0.66
Test Score = 0.62
```

In [468]:
```
## Make a full pipeline with the DecisionTree model as the second step
dt_pipe2 = Pipeline([('text_pipe2',text_pipe),
                     ('clf',DecisionTreeClassifier(max_depth=6,class_we
dt_pipe2
```

executed in 186ms, finished 01:27:15 2022-08-16

Out[468]:

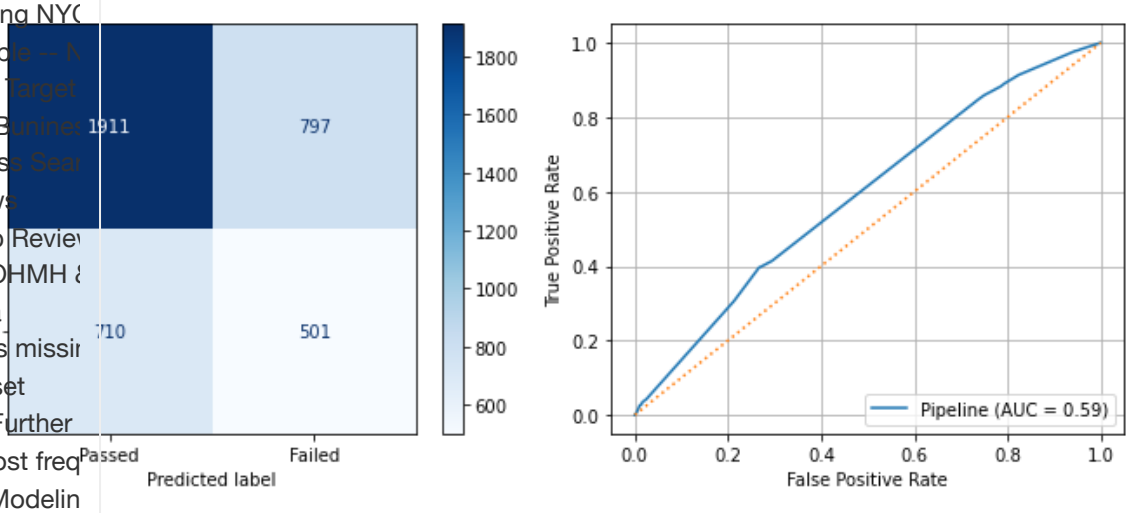


Contents

In [469]:
```
## Modeling with full pipeline
dt_pipe2.fit(X_train,y_train)
evaluate_classification(dt_pipe2,X_test,y_test,X_train=X_train, y_train
```

executed in 1m 6.02s, finished 01:28:23 2022-08-16

```
              precision    recall  f1-score   support

      Passed       0.73      0.71      0.72      2708
      Failed       0.39      0.41      0.40      1211

    accuracy                           0.62      3919
   macro avg       0.56      0.56      0.56      3919
weighted avg       0.62      0.62      0.62      3919
```




```
Training Score = 0.66
Test Score = 0.62
```

The DecisionTree model has shown the best Recall score yet. Not a great accuracy but catching 65% of the businesses that failed their inspections is a 7% improvement from the Logistic Regression classifier.

Hyperparameter Tuning With GridsearchCV

Contents

In [336]:
```
dt_pipe.get_params().keys()
```
executed in 10ms, finished 19:36:04 2022-08-15

Out[336]:
```
dict_keys(['memory', 'steps', 'verbose', 'text_pipe', 'clf', 'text_pipe__
memory', 'text_pipe__steps', 'text_pipe__verbose', 'text_pipe__count_vect
orizer', 'text_pipe__tf_transformer', 'text_pipe__count_vectorizer__analy
zer', 'text_pipe__count_vectorizer__binary', 'text_pipe__count_vectorizer
__decode_error', 'text_pipe__count_vectorizer__dtype', 'text_pipe__count_
vectorizer__encoding', 'text_pipe__count_vectorizer__input', 'text_pipe__
count_vectorizer__lowercase', 'text_pipe__count_vectorizer__max_df', 'tex
t_pipe__count_vectorizer__max_features', 'text_pipe__count_vectorizer__mi
n_df', 'text_pipe__count_vectorizer__ngram_range', 'text_pipe__count_vect
orizer__preprocessor', 'text_pipe__count_vectorizer__stop_words', 'text_p
ipe__count_vectorizer__strip_accents', 'text_pipe__count_vectorizer__toke
n_pattern', 'text_pipe__count_vectorizer__tokenizer', 'text_pipe__count_v
ectorizer__vocabulary', 'text_pipe__tf_transformer__norm', 'text_pipe__tf
_transformer__smooth_idf', 'text_pipe__tf_transformer__sublinear_tf', 'te
xt_pipe__tf_transformer__use_idf', 'clf__ccp_alpha', 'clf__class_weight',
'clf__criterion', 'clf__max_depth', 'clf__max_features', 'clf__max_leaf_n
odes', 'clf__min_impurity_decrease', 'clf__min_impurity_split', 'clf__min
_samples_leaf', 'clf__min_samples_split', 'clf__min_weight_fraction_lea
f', 'clf__presort', 'clf__random_state', 'clf__splitter'])
```

Contents

In [338]:
```
params = {'text_pipe__tf_transformer__use_idf':[True, False],
 'text_pipe__count_vectorizer__tokenizer':[None,tokenizer.toke
 'text_pipe__count_vectorizer__stop_words':[None,stopwords_lis
 'clf__class_weight': ['balanced'],
 'clf__criterion': ['gini','entropy'],
 'clf__max_depth': [4, 6, 10, 12]}

gs = GridSearchCV(estimator=dt_pipe, param_grid = params,
                  scoring='recall',cv=3,n_jobs=-1,verbose=3)

gs.fit(X_train,  y_train)
gs.best_params_
```
executed in 4m 20s, finished 19:43:25 2022-08-15

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  24 out of  24 | elapsed:  4.0min finished
```

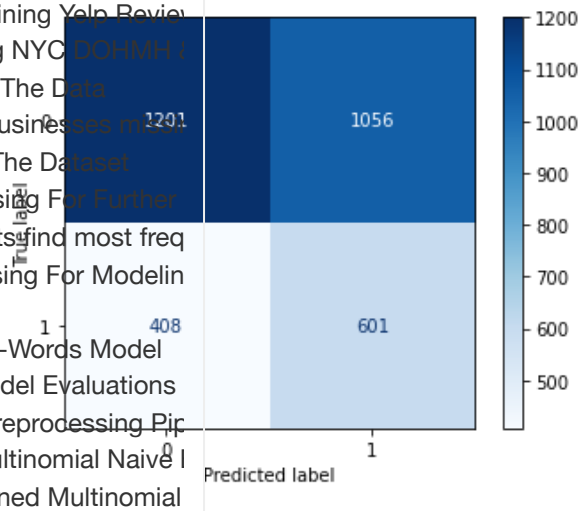
Out[338]:
```
{'clf__class_weight': 'balanced',
 'clf__criterion': 'gini',
 'clf__max_depth': 4}
```

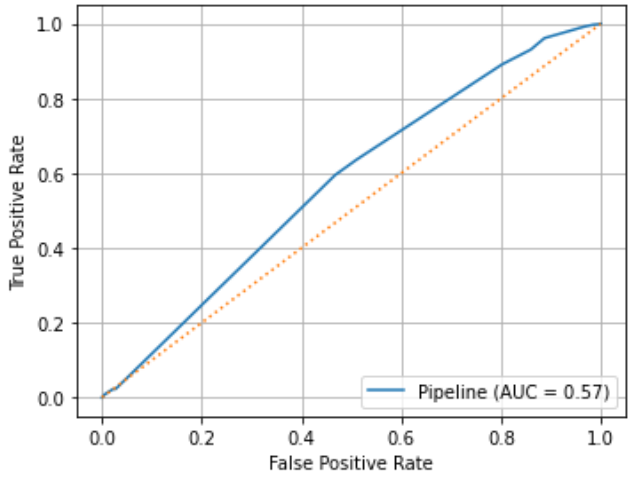
8.2.6 Tuned DecisionTree Model

In [340]:
```
tuned_dt_pipe = Pipeline([('text_pipe',text_pipe),
                          ('clf',DecisionTreeClassifier(max_depth=4,
                                                        criterion='gin
                                                        class_weight='
                          ])

tuned_dt_pipe.fit(X_train,y_train)
evaluate_classification(tuned_dt_pipe,X_test,y_test,X_train=X_train,
                        y_train=y_train)
```

executed in 56.3s, finished 19:46:11 2022-08-15

	precision	recall	f1-score	support
0	0.75	0.53	0.62	2257
1	0.36	0.60	0.45	1009
accuracy			0.55	3266
macro avg	0.55	0.56	0.54	3266
weighted avg	0.63	0.55	0.57	3266






```
Training Score = 0.58
Test Score = 0.55
```

Surprisingly, the cross-validation GridSearch did not improve the models Recall performance.

In [ ]:
```
dt_probas = dt_pipe.predict_proba(X_test)
```

executed in 6.11s, finished 20:50:26 2022-08-15

In [373]:
```
probs_dt = pd.DataFrame(dt_probas, columns= ['Pass', 'Fail'])
probs_dt = pd.concat([probs_dt,y_actual],axis=1)
probs_dt
```

executed in 28ms, finished 20:51:41 2022-08-15

Out[373]:

	Pass	Fail	Severe
0	0.59	0.41	0
1	0.33	0.67	0
2	0.59	0.41	0
3	0.36	0.64	0
4	0.36	0.64	0
...	...	...	...
3261	0.46	0.54	0
3262	0.46	0.54	0
3263	0.75	0.25	0
3264	0.59	0.41	0
3265	0.46	0.54	1

3266 rows × 3 columns

In [402]:
```
dt_pipe.named_steps['clf'].feature_importances_
```

executed in 54ms, finished 21:41:26 2022-08-15

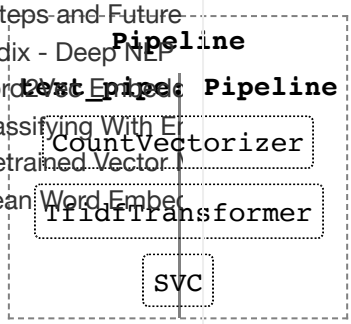
Out[402]:
```
array([0., 0., 0., ..., 0., 0., 0.])
```

8.2.7 Support Vector Classifier Model

In [458]:
```
## Make a full pipeline with the Support Vector Classifier model as the
svc_pipe = Pipeline([('text_pipe',text_pipe),
                     ('clf',SVC(class_weight='balanced'))])
svc_pipe
```

executed in 282ms, finished 22:58:52 2022-08-15

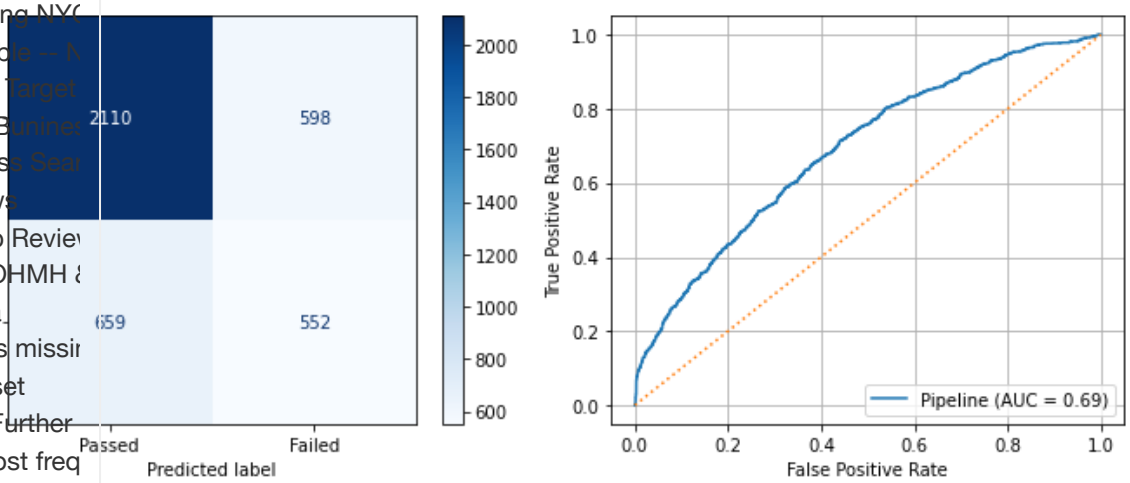
Out[458]:



In [459]:
```
## Modeling with full pipeline
svc_pipe.fit(X_train,y_train)
evaluate_classification(svc_pipe,X_test,y_test,X_train=X_train, y_train
```

executed in 19m 45s, finished 23:18:38 2022-08-15

```
              precision    recall  f1-score   support

      Passed       0.76      0.78      0.77      2708
      Failed       0.48      0.46      0.47      1211

    accuracy                           0.68      3919
   macro avg       0.62      0.62      0.62      3919
weighted avg       0.67      0.68      0.68      3919
```




```
Training Score = 0.96
Test Score = 0.68
```

Not a very good baseline from the SVC model. Recall score of 46% and 68% accuracy. We'll see if tuned parameters make an improvement.

Hyperparameter Tuning With GridsearchCV

Contents

In [460]:
```
1 svc_pipe.get_params().keys()
```
executed in 31ms, finished 23:18:38 2022-08-15

Out[460]:
```
dict_keys(['memory', 'steps', 'verbose', 'text_pipe', 'clf', 'text_pipe__
memory', 'text_pipe__steps', 'text_pipe__verbose', 'text_pipe__count_vect
orizer', 'text_pipe__tf_transformer', 'text_pipe__count_vectorizer__analy
zer', 'text_pipe__count_vectorizer__binary', 'text_pipe__count_vectorizer
__decode_error', 'text_pipe__count_vectorizer__dtype', 'text_pipe__count_
vectorizer__encoding', 'text_pipe__count_vectorizer__input', 'text_pipe__
count_vectorizer__lowercase', 'text_pipe__count_vectorizer__max_df', 'tex
t_pipe__count_vectorizer__max_features', 'text_pipe__count_vectorizer__mi
n_df', 'text_pipe__count_vectorizer__ngram_range', 'text_pipe__count_vect
orizer__preprocessor', 'text_pipe__count_vectorizer__stop_words', 'text_p
ipe__count_vectorizer__strip_accents', 'text_pipe__count_vectorizer__toke
n_pattern', 'text_pipe__count_vectorizer__tokenizer', 'text_pipe__count_v
ectorizer__vocabulary', 'text_pipe__tf_transformer__norm', 'text_pipe__tf
_transformer__smooth_idf', 'text_pipe__tf_transformer__sublinear_tf', 'te
xt_pipe__tf_transformer__use_idf', 'clf__C', 'clf__break_ties', 'clf__cac
he_size', 'clf__class_weight', 'clf__coef0', 'clf__decision_function_shap
e', 'clf__degree', 'clf__gamma', 'clf__kernel', 'clf__max_iter', 'clf__pr
obability', 'clf__random_state', 'clf__shrinking', 'clf__tol', 'clf__verb
ose'])
```

Contents

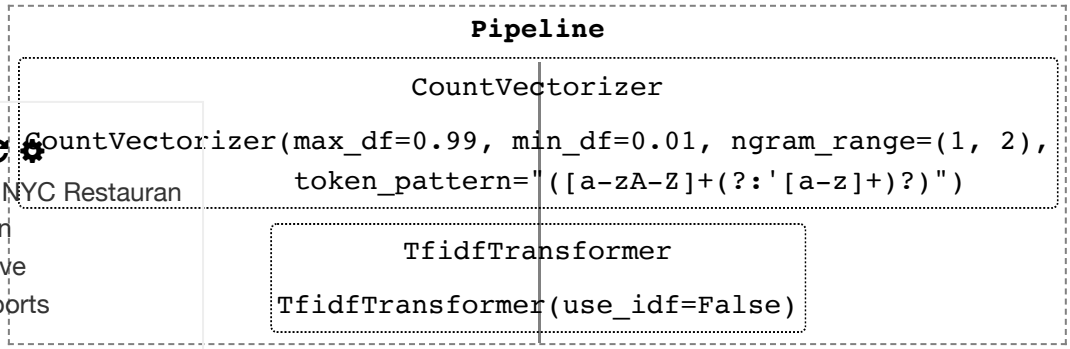
In [462]:
```
1 params = {'text_pipe__tf_transformer__use_idf':[True, False],
2  'text_pipe__count_vectorizer__token_pattern':[None,r"([a-zA-Z
3  'clf__class_weight': ['balanced'],
4  'clf__kernel': ['rbf', 'sigmoid'],
5  'clf__gamma': ['auto','scale']}
6
7 gs = GridSearchCV(estimator=svc_pipe, param_grid = params,
8                    scoring='recall',cv=3,n_jobs=-1,verbose=3)
9
10 gs.fit(X_train,  y_train)
11 gs.best_params_
```
executed in 49m 43s, finished 00:12:46 2022-08-16

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed: 25.9min
[Parallel(n_jobs=-1)]: Done  48 out of  48 | elapsed: 44.2min finished
```

Out[462]:
```
{'clf__class_weight': 'balanced',
 'clf__gamma': 'scale',
 'clf__kernel': 'sigmoid',
 'text_pipe__count_vectorizer__token_pattern': "([a-zA-Z]+(?:'[a-z]+)?)",
 'text_pipe__tf_transformer__use_idf': False}
```

In [463]:
```
1 text_pipe2
```
executed in 194ms, finished 00:32:52 2022-08-16

Out[463]:



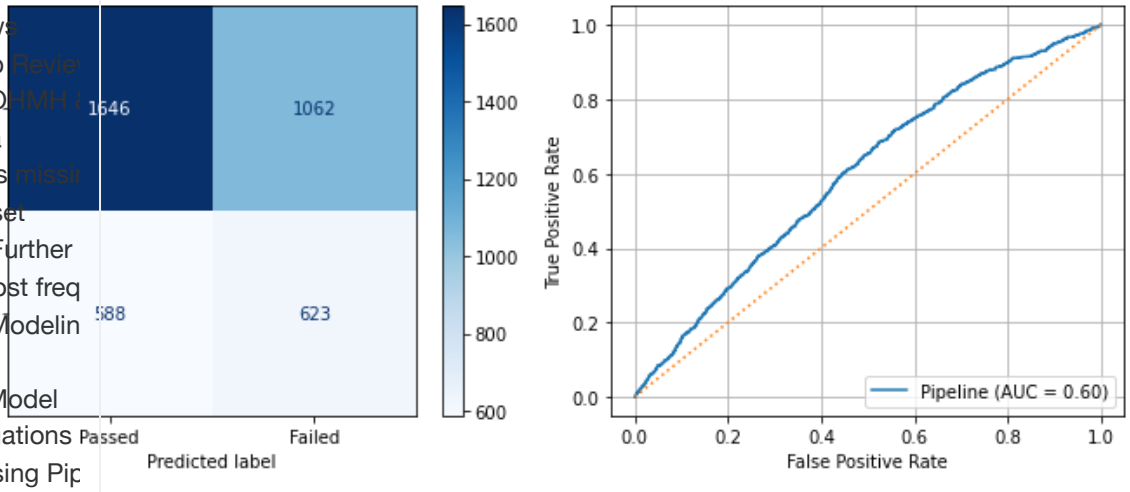
Contents

8.2.8 Tuned SupportVectorClassifier Model

In [465]:
```
tuned_svc_pipe = Pipeline([('text_pipe2',text_pipe2),
                           ('clf',SVC(gamma='scale',kernel='sigmoid',
                                      class_weight='balanced'))
                          ])

tuned_svc_pipe.fit(X_train,y_train)
evaluate_classification(tuned_svc_pipe,X_test,y_test,X_train=X_train,
                        y_train=y_train)
```

executed in 40m 2s, finished 01:14:34 2022-08-16

	precision	recall	f1-score	support
Passed	0.74	0.61	0.67	2708
Failed	0.37	0.51	0.43	1211
accuracy			0.58	3919
macro avg	0.55	0.56	0.55	3919
weighted avg	0.62	0.58	0.59	3919




```
Training Score = 0.61
Test Score = 0.58
```

Model results

9 Conclusions

9.1 Best Model Results

In [ ]:
```

```

9.2 Next Steps and Future Work

Data Source

Hyperparameter Tuning

Neural Network Training

9.3 Appendix - Deep NLP Models

After fitting and tuning a handful of bag-of-n-grams, we can try solving the business problem with a simple neural network. First we will need to generate word embeddings. Word embeddings map each word to one vector, which can then be learned in a way which resembles a neural network. Embedding vectors differ from bag-of-n-grams because they try to capture various characteristics of that word with regard to the overall text. To get our word embeddings, we will be using Word2Vec.

The simple bag-of-words featurized models above summarize a document, in this case restaurant reviews, by the words present in that review and ignores their context. Below we will attempt to take context into account using Recurrent Neural Networks (RNNs) that treat the doument as a sequence, taking account of all the words in the context of those preceded and following.

9.3.1 Word2Vec Embeddings

Word2Vec CBOW

In [477]:
```
# Minimally preprocessing for Deep NLP with all lower case
# Keeping stopwords to help provide firther word context
lower_data = model_df['Reviews'].map(lambda x: simple_preprocess(x.lowe
                                                                 deacc=
```

executed in 1m 1.44s, finished 08:09:36 2022-08-16

In [478]:
```
# CBOW (Continuous Bag of Words)
w2v_cbow_model = Word2Vec(lower_data,size=50,window=3,min_count=3,sg=0)
```

executed in 5m 12s, finished 08:14:58 2022-08-16

In [480]:
```
w2v_cbow_model
```

executed in 48ms, finished 08:15:25 2022-08-16

Out[480]:
```
<gensim.models.word2vec.Word2Vec at 0x7fc8a5ee8340>
```

In [481]:
```
examples = w2v_cbow_model.corpus_count
```

executed in 26ms, finished 08:15:54 2022-08-16

In [482]:
```
# Training Word2Vec Model
w2v_cbow_model.train(lower_data,total_examples=examples,epochs=50)
```

executed in 9m 44s, finished 08:25:42 2022-08-16

Out[482]:
```
(464260994, 610253300)
```

Contents

In [196]:
```
# Get the keyed vector
wv = w2v_cbow_model.wv
len(wv.vocab)
```
executed in 34ms, finished 13:31:11 2022-08-12

Out[196]: 33298

Contents

We have 33,298 trained words.

In [197]:
```
# Cosine similarity of embedded word vectors
wv.most_similar(negative=['sick'])
```
executed in 69ms, finished 13:31:11 2022-08-12

Out[197]:
```
[('crudite', 0.5211197733879089),
 ('offset', 0.5062964558601379),
 ('serviceprice', 0.49563947319984436),
 ('is', 0.4898544251918793),
 ('offers', 0.48499834537506104),
 ('albeit', 0.4783792197704315),
 ('quant', 0.4731408953666687),
 ('mexodus', 0.47181084752082825),
 ('includes', 0.47130322456359863),
 ('namely', 0.4589795768260956)]
```

In [198]:
```
wv.most_similar(negative=['poison'])
```
executed in 16ms, finished 13:31:11 2022-08-12

Out[198]:
```
[('pere', 0.5644559860229492),
 ('yogurty', 0.5012726783752441),
 ('doughier', 0.4855877161026001),
 ('waaayyyy', 0.48042428493499756),
 ('slightly', 0.47897931933403015),
 ('crispy', 0.4708661125946045),
 ('crisp', 0.4691073000431061),
 ('deconstructed', 0.4550844132900238),
 ('buckys', 0.45102351903915405),
 ('chilly', 0.4506188631057739)]
```

In [199]:
```
wv.most_similar(negative=['health'])
```
executed in 14ms, finished 13:31:11 2022-08-12

Out[199]:
```
[('skinny', 0.5029502511024475),
 ('goodworth', 0.4860212206840515),
 ('petite', 0.4592294991016388),
 ('plasticy', 0.44577306509017944),
 ('sweet', 0.44104284048080444),
 ('towelettes', 0.43190431594848633),
 ('bedrock', 0.42919042706489563),
 ('tart', 0.4178142547607422),
 ('gooood', 0.4148668050765991),
 ('discombobulated', 0.41178783774375916)]
```

Word2Vec Skip-Gram

In [477]:
```
# Minimally preprocessing for Deep NLP with all lower case
# Keeping stopwords to help provide firther word context
lower_data = model_df['Reviews'].map(lambda x: simple_preprocess(x.lowe
                                                                 deacc=
```

In [483]:
```
# SG (Skip Gram)
w2v_sg_model = Word2Vec(lower_data,size=50,window=3,min_count=3,sg=1)
```
executed in 2m 35s, finished 08:33:06 2022-08-16

In [484]:
```
w2v_sg_model
```
executed in 16ms, finished 08:33:06 2022-08-16

Out[484]: <gensim.models.word2vec.Word2Vec at 0x7fc9bc8f96a0>

In [485]:
```
examples = w2v_sg_model.corpus_count
```
executed in 4ms, finished 08:33:06 2022-08-16

In [487]:
```
#Training Word2Vec Model
w2v_sg_model.train(lower_data,total_examples=examples,epochs=50)
```
executed in 21m 32s, finished 08:54:38 2022-08-16

Out[487]: (464235054, 610253300)

In [488]:
```
# Get the keyed vector
wv = w2v_sg_model.wv
len(wv.vocab)
```
executed in 17ms, finished 08:54:39 2022-08-16

Out[488]: 33298

Again, there are 33,298 trained words.

In [489]:
```
wv.most_similar(negative=['sick'])
```
executed in 219ms, finished 08:54:39 2022-08-16

Out[489]:
```
[('lends', 0.3013019561767578),
 ('aquatic', 0.2592369616031647),
 ('coffeehouse', 0.24427442252635956),
 ('becherovka', 0.24270877242088318),
 ('reduction', 0.24244828522205353),
 ('beaubourg', 0.24152667820453644),
 ('designer', 0.2410590648651123),
 ('fabrics', 0.2288820892572403),
 ('every', 0.22802232205867767),
 ('contains', 0.22213214635849)]
```

Contents

In [490]:
```
wv.most_similar(negative=['poison'])
```
executed in 25ms, finished 08:54:39 2022-08-16

Out[490]:
```
[('formally', 0.3024853765964508),
 ('slightly', 0.2836840748786926),
 ('zabzi', 0.28303104639053345),
 ('hilltop', 0.27307450771331787),
 ('brat', 0.2700038254261017),
 ('deconstructed', 0.2691287100315094),
 ('edition', 0.25647905468940735),
 ('tad', 0.25589582324028015),
 ('dolce', 0.2506220042705536),
 ('wrongly', 0.24508269131183624)]
```


In [491]:
```
wv.most_similar(negative=['health'])
```
executed in 38ms, finished 08:54:39 2022-08-16

Out[491]:
```
[('crunched', 0.29327139258384705),
 ('tacha', 0.25872737169265747),
 ('creston', 0.24874328076839447),
 ('yamria', 0.24648892879486084),
 ('moons', 0.21890901029109955),
 ('imagined', 0.21743138134479523),
 ('snickers', 0.21641209721565247),
 ('tart', 0.2162439078092575),
 ('fella', 0.21612727642059326),
 ('kunafa', 0.20877067744731903)]
```

9.3.2 Classifying With Embeddings

In [494]:
```
X = model_df['Reviews']
y = model_df['Severe']
y.tail()
```
executed in 43ms, finished 09:09:45 2022-08-16

Out[494]:
```
13056    1
13057    1
13058    0
13059    0
13060    0
Name: Severe, dtype: int64
```

In [495]:
```
# Train Test Split before any pre-processing
X_train_wv, X_test_wv, y_train_wv, y_test_wv = train_test_split(X,y,ran
X_train_wv.shape, y_test_wv.shape
```
executed in 56ms, finished 09:09:47 2022-08-16

Out[495]:
```
((9795,), (3266,))
```

In [496]:
```
pd.Series(y_test).value_counts(normalize=True)
```
executed in 332ms, finished 09:09:49 2022-08-16

Out[496]:
```
0    0.69
1    0.31
Name: Severe, dtype: float64
```

Contents

In [497]:
```
# Class imbalance
y_train.value_counts(1)
```
executed in 24ms, finished 09:09:50 2022-08-16

Out[497]:
```
0    0.71
1    0.29
Name: Severe, dtype: float64
```

In [234]:
```
#Calculating class weights

weights = compute_class_weight('balanced',np.unique(y_train), y_train)
weights_dict = dict(zip( np.unique(y_train),weights))
weights_dict
```
executed in 57ms, finished 18:39:52 2022-08-13

Out[234]:
```
{0: 0.7096797565570208, 1: 1.692294402211472}
```

In [235]:
```
# OneHotEncoding target variable for NN
y_train_seq = to_categorical(y_train)
y_test_seq = to_categorical(y_test)
y_train_seq.shape
```
executed in 28ms, finished 18:39:52 2022-08-13

Out[235]:
```
(9785, 2)
```

In [499]:
```
# Tokenizing text with Keras tokenizer
max_words = 30000
tokenizer = text.Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X_train)
train_sequences = tokenizer.texts_to_sequences(X_train)
test_sequences = tokenizer.texts_to_sequences(X_test)
```
executed in 447ms, finished 09:17:06 2022-08-16

In [501]:
```
# Finding maximum sequence length for optimaiztion
seq_lenghts = list(map(lambda x: len(x),[*train_sequences,*test_sequenc

```
executed in 29ms, finished 09:18:14 2022-08-16

Out[501]:
```
10700
```

In [505]:
```
# Finding average sequence length for optimaiztion
import statistics
statistics.mean(seq_lenghts)
```
executed in 59ms, finished 09:22:17 2022-08-16

Out[505]:
```
991.4490467804916
```

In [506]:
```
max_seq_length = 2000
```
executed in 5ms, finished 09:23:14 2022-08-16

In [507]:
```
# Padding sequences because NN require same sizes
X_train_seq = keras.preprocessing.sequence.pad_sequences(train_sequence
                                                         maxlen=max_sequence_length)
X_test_seq = keras.preprocessing.sequence.pad_sequences(test_sequences,
                                                        maxlen=max_sequence_length)
```
executed in 2.88s, finished 09:23:28 2022-08-16

In [508]:
```
len(tokenizer.index_word)
```
executed in 24ms, finished 09:23:50 2022-08-16

Out[508]:
```
75799
```

In [509]:
```
# Adapted from https://nbviewer.org/github/learn-co-curriculum/aug-ds-f
def get_earlystop(monitor='val_accuracy',patience=3, restore_best_weigh
    """"""
    args = locals()
    return EarlyStopping(**args)

get_earlystop.__doc__+=EarlyStopping.__doc__
get_earlystop
```
executed in 64ms, finished 09:26:23 2022-08-16

Out[509]:
```
<function __main__.get_earlystop(monitor='val_accuracy', patience=3, rest
ore_best_weights=False)>
```

In [243]:
```
def make_model1(EMBEDDING_SIZE = 128):
    model=Sequential()

    model.add(Embedding(max_words, EMBEDDING_SIZE))
    model.add(LSTM(50,return_sequences=False))
    model.add(Dropout(0.5))
    model.add(Dense(25, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(2, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    #     display(model.summary())
    return model
```
executed in 24ms, finished 18:40:28 2022-08-13

Contents

In [245]:
```
model = make_model1()
history = model.fit(X_train_seq, y_train_seq, epochs=5,
                    batch_size=32, validation_split=0.2,callbacks=get_e
                   class_weight=weights_dict)
```

executed in 5h 44m 56s, finished 00:25:42 2022-08-14

```
Epoch 1/5
245/245 [==============================] - 4003s 16s/step - loss: 0.6945
- accuracy: 0.4777 - val_loss: 0.6907 - val_accuracy: 0.5666
Epoch 2/5
245/245 [==============================] - 4307s 18s/step - loss: 0.6707
- accuracy: 0.5972 - val_loss: 0.7343 - val_accuracy: 0.4844
Epoch 3/5
245/245 [==============================] - 4273s 17s/step - loss: 0.5333
- accuracy: 0.7389 - val_loss: 0.7619 - val_accuracy: 0.5426
Epoch 4/5
245/245 [==============================] - 4153s 17s/step - loss: 0.3294
- accuracy: 0.8647 - val_loss: 0.8504 - val_accuracy: 0.6064
Epoch 5/5
245/245 [==============================] - 3952s 16s/step - loss: 0.1898
- accuracy: 0.9343 - val_loss: 1.1274 - val_accuracy: 0.6110
```

In [251]:
```
model.summary()
```

executed in 126ms, finished 15:55:08 2022-08-14

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 128)         6400000
_________________________________________________________________
lstm_1 (LSTM)                (None, 50)                35800
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 25)                1275
_________________________________________________________________
dropout_3 (Dropout)          (None, 25)                0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 52
=================================================================
Total params: 6,437,127
Trainable params: 6,437,127
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
# Collapsing y_test data down to single column
y_test_seq.argmax(axis=1)
```

executed in 3m 6s, finished 09:42:51 2022-08-16

In [252]:
```
# model.save('First_LSTM_Model.h5')
```

executed in 631ms, finished 16:03:15 2022-08-14

Contents

In [519]:
```
new_model = load_model('First_LSTM_Model.h5')
```
executed in 2.03s, finished 09:52:03 2022-08-16

In [520]:
```
new_model
```
executed in 11ms, finished 09:52:10 2022-08-16

Out[520]: <keras.engine.sequential.Sequential at 0x7fc8c31af340>

In [524]:
```
#  Collapsing down to which class is predicted
y_hat_test = new_model.predict(X_test_seq).argmax(axis=1)
y_hat_test[:5]
```
executed in 3m 33s, finished 09:59:48 2022-08-16

```
123/123 [==============================] - 212s 2s/step
```

Out[524]: array([0, 0, 0, 1, 1])

In [249]:
```
# # Tokenizing text with Keras tokenizer
# max_words2 = 25000
# tokenizer2 = text.Tokenizer(num_words=max_words)
```
executed in 10ms, finished 15:12:22 2022-08-14

In [250]:
```
# def make_model2(EMBEDDING_SIZE = 128):
#     model=Sequential()

#     model.add(Embedding(max_words, EMBEDDING_SIZE))
#     model.add(LSTM(50,return_sequences=False))
#     # model.add(GlobalMaxPool1D())
#     model.add(Dropout(0.5))
#     model.add(Dense(25, activation='relu'))
#     model.add(Dropout(0.5))
#     model.add(Dense(2, activation='softmax'))

#     model.compile(loss='categorical_crossentropy',
#                   optimizer='adam',
#                   metrics=['accuracy'])
#     # display(model.summary())
#     return model
```
executed in 7ms, finished 15:12:23 2022-08-14

Contents

In [245]:
```
# model2 = make_model2()
# history = model.fit(X_train_seq, y_train_seq, epochs=10,
#                     batch_size=32, validation_split=0.2,callbacks=get
#                    class_weight=weights_dict)
```

executed in 5h 44m 56s, finished 00:25:42 2022-08-14

```
Epoch 1/5
245/245 [==============================] - 4003s 16s/step - loss: 0.6945
- accuracy: 0.4777 - val_loss: 0.6907 - val_accuracy: 0.5666
Epoch 2/5
245/245 [==============================] - 4307s 18s/step - loss: 0.6707
- accuracy: 0.5972 - val_loss: 0.7343 - val_accuracy: 0.4844
Epoch 3/5
245/245 [==============================] - 4273s 17s/step - loss: 0.5333
- accuracy: 0.7389 - val_loss: 0.7619 - val_accuracy: 0.5426
Epoch 4/5
245/245 [==============================] - 4153s 17s/step - loss: 0.3294
- accuracy: 0.8647 - val_loss: 0.8504 - val_accuracy: 0.6064
Epoch 5/5
245/245 [==============================] - 3952s 16s/step - loss: 0.1898
- accuracy: 0.9343 - val_loss: 1.1274 - val_accuracy: 0.6110
```

61% Validation accuracy score is not much better performance than the Bag-of-N-gram models we tried first.

In [528]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

executed in 42ms, finished 10:01:19 2022-08-16

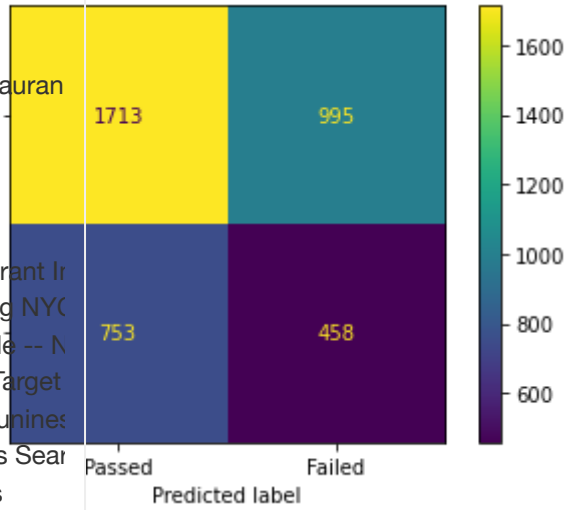
In [533]:
```
cm = confusion_matrix(y_test_seq.argmax(axis=1),y_hat_test)
```

executed in 14ms, finished 10:02:54 2022-08-16

Contents

In [540]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=['Pass
disp.plot()
plt.show()
print(classification_report(y_test_seq.argmax(axis=1),y_hat_test))
```

executed in 733ms, finished 10:08:00 2022-08-16




```
              precision    recall  f1-score   support

           0       0.69      0.63      0.66      2708
           1       0.32      0.38      0.34      1211

    accuracy                           0.55      3919
   macro avg       0.50      0.51      0.50      3919
weighted avg       0.58      0.55      0.56      3919
```

In [ ]:
```
1
```

9.3.3 Pretrained Vector Models

Our Embeddings

In [542]:
```
1 wv.get_keras_embedding()
```

executed in 116ms, finished 10:11:24 2022-08-16

Out[542]: `<keras.layers.core.embedding.Embedding at 0x7fc8640ecf40>`

Contents

In [545]:
```
def make_model_wv(wv):
    model=Sequential()

    model.add(wv.get_keras_embedding())
    model.add(LSTM(50,return_sequences=False))
    model.add(Dense(25, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(2, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    return model
```

executed in 12ms, finished 10:15:39 2022-08-16

In [ ]:
```

```

In [546]:
```
model3 = make_model_wv(wv)
history = model3.fit(X_train_seq, y_train_seq, epochs=5,
                     batch_size=32, validation_split=0.2,
                     callbacks=get_earlystop(),class_weight=weights_dict
```

executed in 1h 50m 41s, finished 12:06:22 2022-08-16

```
Epoch 1/5
229/229 [==============================] - 1215s 5s/step - loss: 0.6946 -
accuracy: 0.5105 - val_loss: 0.6881 - val_accuracy: 0.6320
Epoch 2/5
229/229 [==============================] - 1300s 6s/step - loss: 0.6938 -
accuracy: 0.5143 - val_loss: 0.6983 - val_accuracy: 0.3461
Epoch 3/5
229/229 [==============================] - 1534s 7s/step - loss: 0.6924 -
accuracy: 0.4925 - val_loss: 0.6835 - val_accuracy: 0.6402
Epoch 4/5
229/229 [==============================] - 1327s 6s/step - loss: 0.6907 -
accuracy: 0.5263 - val_loss: 0.6785 - val_accuracy: 0.6337
Epoch 5/5
229/229 [==============================] - 1264s 6s/step - loss: 0.6881 -
accuracy: 0.5479 - val_loss: 0.6792 - val_accuracy: 0.5992
```

In [551]:
```
y_hat_test = model.predict(X_test_seq).argmax(axis=1)
```

executed in 3m 48s, finished 12:15:37 2022-08-16

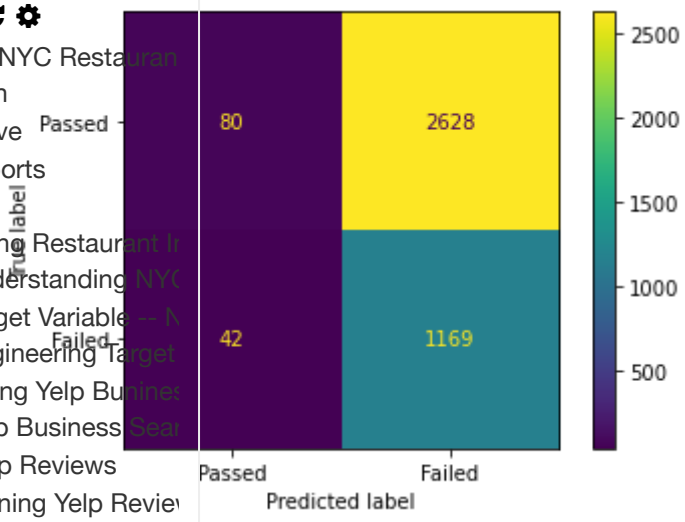
```
123/123 [==============================] - 224s 2s/step
```

Contents

In [556]:
```
cm = confusion_matrix(y_test_seq.argmax(axis=1),y_hat_test)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Pas
disp.plot()
plt.show()
print(classification_report(y_test_seq.argmax(axis=1),y_hat_test))
```

executed in 432ms, finished 12:21:16 2022-08-16

Contents



	precision	recall	f1-score	support
0	0.66	0.03	0.06	2708
1	0.31	0.97	0.47	1211
accuracy			0.32	3919
macro avg	0.48	0.50	0.26	3919
weighted avg	0.55	0.32	0.18	3919

Pretrained Embeddings With GLOVE

In [547]:
```
target = model_df['Severe']
```

executed in 193ms, finished 12:06:22 2022-08-16

In [548]:
```
data = model_df['Reviews'].map(word_tokenize).values
```

executed in 3m 8s, finished 12:09:30 2022-08-16

In [549]:
```
total_vocabulary = set(word for review in data for word in review)
```
executed in 2.19s, finished 12:09:32 2022-08-16

In [550]:
```
len(total_vocabulary)
print('There are {} unique tokens in the dataset.'.format(len(total_voc
```
executed in 36ms, finished 12:09:32 2022-08-16

```
There are 180157 unique tokens in the dataset.
```

Contents

In [243]:
```
glove = {}
with open('glove.6B.50d.txt', 'rb') as f:
    for line in f:
        parts = line.split()
        word = parts[0].decode('utf-8')
        if word in total_vocabulary:
            vector = np.array(parts[1:], dtype=np.float32)
            glove[word] = vector
```
executed in 2.43s, finished 14:50:31 2022-08-12

In [244]:
```
# testing vecotrizer
glove['food']
```
executed in 40ms, finished 14:50:33 2022-08-12

Out[244]:
```
array([-0.47222  , -0.44545  , -0.51833  , -0.26818  ,  0.44427  ,
       -0.25108  , -0.99282  , -0.90198  ,  1.8729   ,  0.039081 ,
        0.14284  ,  0.074878 ,  1.0543   , -0.3203   ,  1.0722   ,
        0.44323  ,  0.0099484,  0.15754  ,  0.51399  , -0.77668  ,
        0.924    ,  0.010958 ,  0.58815  ,  0.23078  , -0.34281  ,
       -0.88444  , -0.31492  ,  0.12661  ,  1.1445   ,  0.60775  ,
        3.4344   ,  0.63561  , -0.13832  ,  0.28045  , -0.16181  ,
        0.77541  , -0.49888  ,  0.4602   ,  0.91799  ,  0.29007  ,
        0.06884  ,  0.59978  ,  0.53967  , -0.061752 ,  1.2975   ,
        0.92323  , -0.80945  ,  0.34932  ,  0.33934  ,  0.25499  ],
      dtype=float32)
```

9.3.4 Mean Word Embeddings

In [245]:
```
# # Adapted from https://github.com/learn-co-curriculum/dsc-classificat
# class W2vVectorizer(object):

#     def __init__(self, w2v):
#         # Takes in a dictionary of words and vectors as input
#         self.w2v = w2v
#         if len(w2v) == 0:
#             self.dimensions = 0
#         else:
#             self.dimensions = len(w2v[next(iter(glove))])

#     def fit(self, X, y):
#         return self

#     def transform(self, X):
#         return np.array([
#             np.mean([self.w2v[w] for w in words if w in self.w2v]
#                     or [np.zeros(self.dimensions)], axis=0) for words
```
executed in 8ms, finished 14:50:35 2022-08-12

Passing the mean vectorizer class created above in the first step of pipeline. To be followed it up with the model classifying the data fed in.

In [246]:
```
# Creating pipeline objects that make use of the mean embedding vectori
rf = Pipeline([('Word2Vec Vectorizer', W2vVectorizer(glove)),
               ('DecisionTree', DescisionTreeClassifier() verbose=True))
svc = Pipeline([('Word2Vec Vectorizer', W2vVectorizer(glove)),
                ('Support Vector Machine', SVC())])
lr = Pipeline([('Word2Vec Vectorizer', W2vVectorizer(glove)),
               ('Logistic Regression', LogisticRegression())])
```
executed in 88ms, finished 14:50:37 2022-08-12

In [247]:
```
# Saving a list for each pipeline name and object
models = [('Random Forest', rf),
          ('Support Vector Machine', svc),
          ('Logistic Regression', lr)]
```
executed in 3ms, finished 14:50:39 2022-08-12

Contents

In [248]:
```
# Storing cross validation scores with sklearn's cross_val_score() func
cv_scores = [(name, cross_val_score(model, data, target, cv=2).mean())
cv_scores
```

executed in 1m 39.0s, finished 14:52:19 2022-08-12

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    4.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    5.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.1s finished
```

Out[248]:
```
[('Random Forest', 0.6923663864387608),
 ('Support Vector Machine', 0.7011714304941705),
 ('Logistic Regression', 0.7004823502846478)]
```

Contents