In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from random import sample
```

In [2]:
```
import statsmodels.api as sm
```

In [3]:
```
db_biz = pd.read_json("yelp_dataset/business.json", lines=True)
```

In [4]:
```
db_rev = pd.read_json("yelp_dataset/review.json", lines = True)
```

In [5]:
```
#merge relevant columns
piece1 = db_rev[['business_id','text','stars']]
piece2 = db_biz[['business_id','state','categories']]
merged = pd.merge(left=piece1,right=piece2)
```

In [6]:
```
#helper functions
def isRestaurant(categories):
    include = False
    try:
        if 'Restaurants' in categories:
            include = True
        return include
    except:
        return include

def remove_stop_words(article):
    tokenizer = RegexpTokenizer(r'\w+')
    stop_words = set(stopwords.words('english'))

    words = tokenizer.tokenize(article)
    words_without_stop_words = ["" if word in stop_words else word for word in words]
    article_clean = " ".join(words_without_stop_words).strip()
    return article_clean
```

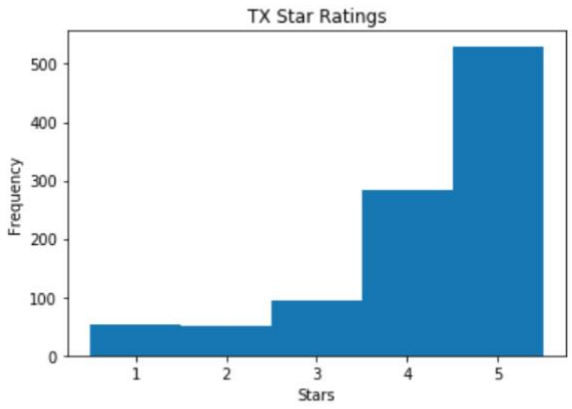
In [7]:
```
#get only restaurants
merged['restaurants'] = merged['categories'].apply(lambda x: True if isRestaurant(x)==True else 'False')
```

In [8]:
```
#only restaurants from Texas
merged_restaurants = merged[(merged['restaurants'] == True)]
merged_r = merged_restaurants[(merged_restaurants['state'] == 'TX')]
```

In [9]:
```
#check to see if any strange values
listy = merged_r['stars']
set(listy)
```

Out[9]:
```
{1, 2, 3, 4, 5}
```

In [10]:
```
#EDA pt. 1- distribution of stars
plot_all = listy.plot.hist(bins = 5,range = [1,6],align = 'left')
plot_all.set_xlabel('Stars')
plot_all.set_ylabel('Frequency')
plot_all.title.set_text('TX Star Ratings')
```



In [11]:
```
from nltk.tokenize import sent_tokenize, word_tokenize, RegexpTokenizer
from nltk.corpus import stopwords
```

In [12]:
```
#separate 5 star reviews
good_stars = merged_r[(merged_r['stars'] > 4)]
bad_stars = merged_r[(merged_r['stars'] <= 4)]
```

In [13]:
```
X = good_stars[['text','state']]
y = good_stars[['stars']]
review_text = list(X['text'])
```

In [15]:
```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
review_text_counts = count_vect.fit_transform(cleaned_review_text)
```

In [18]:
```
r = pd.DataFrame(review_text_counts.toarray(), columns = count_vect.get_feature_names())
```

In [19]:
```
r.shape
```

Out[19]: (530, 4698)

In [24]:
```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
review_text_vectorized = tfidf_transformer.fit_transform(review_text_counts)
```

In [25]:
```
r = review_text_vectorized.toarray()
```

In [26]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

In [27]:
```
#vectorize
mystopwords = set(stopwords.words('english'))
cv=CountVectorizer(max_df=0.85,stop_words=mystopwords)
```

In [28]:
```
word_count_vector=cv.fit_transform(review_text)
word_count_vector.shape
```

Out[28]: (530, 4600)

In [29]:
```
list(cv.vocabulary_.keys())
```

Out[29]:
```
['awesome',
 'place',
 'total',
 'experience',
 'many',
 'times',
 'time',
 'walk',
 'away',
 'pleasant',
 'memories',
 'whether',
 'food',
 'service',
 'try',
 'visit',
 'texas',
 'de',
 'brazil',
 'vegas',
```

In [30]:
```
from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer=TfidfTransformer(smooth_idf=True,use_idf=True)
tfidf_transformer.fit(word_count_vector)
```

Out[30]: TfidfTransformer(norm='l2', smooth_idf=True, sublinear_tf=False, use_idf=True)

In [31]:
```
tf_idf_vector=tfidf_transformer.transform(cv.transform(review_text))
```

In [32]:
```
def sort_coo(coo_matrix):
    tuples = zip(coo_matrix.col, coo_matrix.data)
    return sorted(tuples, key=lambda x: (x[1], x[0]), reverse=True)
```

In [33]:
```
sorted_items=sort_coo(tf_idf_vector.tocoo())
```

In [38]:
```
def extract_topn_from_vector(feature_names, sorted_items, topn=10):
    """get the feature names and tf-idf score of top n items"""

    #use only topn items from vector
    sorted_items = sorted_items[:topn]

    score_vals = []
    feature_vals = []

    # word index and corresponding tf-idf score
    for idx, score in sorted_items:

        #keep track of feature name and its corresponding score
        score_vals.append(round(score, 3))
        feature_vals.append(feature_names[idx])

    #create a tuples of feature,score
    #results = zip(feature_vals,score_vals)
    results= {}
    for idx in range(len(feature_vals)):
        results[feature_vals[idx]]=score_vals[idx]

    return results
```

In [39]:
```
feature_names = cv.get_feature_names()
```

In [37]:
```
keywords=extract_topn_from_vector(feature_names,sorted_items,20)
keywords
```

Out[37]:
```
{'10': 0.764,
 'must': 0.733,
 'qqb': 0.669,
 'fun': 0.652,
 'hands': 0.63,
 'melt': 0.612,
 'five': 0.611,
 'pricy': 0.611,
 'war': 0.603,
 'whenever': 0.593,
 'skip': 0.592,
 'kristen': 0.591,
 'breast': 0.567,
 'definetly': 0.566,
 'yes': 0.565,
 'fast': 0.565,
 'week': 0.558,
 'thumbs': 0.557,
 'number': 0.557}
```

In the code above, we showed two EDA techniques. The first is the histogram showing the distribution of stars in Texas restaurant reviews. The second is the vectorization and TF/IDF technique where we found the words with the highest TF/IDF scores in the pre-existing 5-star reviews.