Business Understanding

An `Airbnb` is a community based platform for `listing` and `renting` local homes that connects hosts and travellers by facilitating the process of renting without owning rooms. It cultivates a sharing-economy since it allows property owners to rent out private flats. This research aims to better understand what factors are considered when an individual chooses to book an Airbnb and what features contribute most to their experience. Established in 2008, Airbnb has experienced growth in the number of rental listings available and it continues to disrupt the hospitality industry with its service offerings. It has helped guests and hosts to travel in a more unique and personalized way. The company went from a single air mattress for rent to global cooperation valued at more than **30 billion dollars** all thanks to its energetic founder- Brian Chesky. Sentiment analysis is extremely important because it helps businesses quickly understand the overall opinions of their customers. By automatically sorting the sentiment behind reviews, businesses can effectively gauge brand reputation, understand customers and make faster and more accurate decisions. Reviews are extremely important on Airbnb as customers are generally wary of airbnbs with bad reviews, while good reviews will increase the number of bookings you get as a host. This study will build from the data to identify a set of broad themes that characterize the attributes that influence Airbnb users' experience in Seattle.

Problem Statement

When choosing an Airbnb, apart from the obvious requirements like price and location, customers tend to spend time reading through guest reviews to understand more about the host and the experience they can expect while staying there. The only problem is that this manual effort can be very time consuming. The main goal of this project is to come up with a way guests can get a concise understanding of prior guests experience without having to read through pages of reviews. Customers are not only interested in knowing whether most reviews were positive they are also interested in knowing what most guests have said about their experience. With this problem framed, the study aims to approach the problem by relevant keyword extraction using TF-IDF (Term Frequency — Inverse Document Frequency) and Text summarizations.

Specific Objectives

- To identify accommodation attributes Airbnb guests use to rate their experience
- To extract sentiments from unstructured customer review texts.
- To build a word cloud with key word attributes customers use in their reviews.

Business Success Criteria

Perform sentiment analysis on reviews of comments left by customers and predicting the given scores based on the reviews displayed in the dataset. Produce snapshots (word cloud) of feedback for airbnbs to allow travellers to compare different options at a glance and make the best choice in no time. Recommend solutions that can benefit hotel owners, online travel agencies, booking sites and travel review platforms seeking ways to put their customers in more relaxed mood.

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt
import seaborn as sns
import re
import string

# Import all my NLTK libraries for stuff
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import nltk
nltk.download(['punkt', 'wordnet', 'stopwords'])
from nltk.tokenize import word_tokenize, sent_tokenize
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer
from sklearn.ensemble import RandomForestRegressor,
RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import
mean_squared_error,accuracy_score,plot_confusion_matrix,
classification_report
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')
from nltk.corpus import stopwords
from nltk.probability import FreqDist
from wordcloud import WordCloud
from wordcloud import STOPWORDS
import tensorflow as tf
from tensorflow.keras.optimizers import SGD,Adam
from tensorflow.keras.constraints import MaxNorm
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM,Dense, Dropout,
SpatialDropout1D,Flatten
from tensorflow.keras.layers import Embedding
from keras.callbacks import ModelCheckpoint
from keras import regularizers
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
[nltk_data] Downloading package punkt to /Users/admin/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/admin/nltk_data...
```

```
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /Users/admin/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```python
pd.set_option('display.max_colwidth', None)

def Uniq_Value(data,col):
    return len(data[col].unique())

def missing_values(data):
    miss_vals = data.isnull().sum().sort_values(ascending=False)

    #percentages
    percentages = (((data.isnull().sum()) /
len(data)).sort_values(ascending=False))*100

    #create dataframe of missing values

    missing_df = pd.DataFrame({"Total missing values": miss_vals,
'Percentage(%)':percentages})

    #if percentage == 0 implies no missing values
    missing_df.drop(missing_df[missing_df['Percentage(%)']==0].index,
inplace = True)

    return missing_df



def clean_text_round1(text):
    '''Make text lowercase, remove text in square brackets, remove
punctuation and remove words containing numbers.'''
    text = text.lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text

round1 = lambda x: clean_text_round1(x)

# Apply a second round of cleaning
def clean_text_round2(text):
    '''Get rid of some additional punctuation and non-sensical text
that was missed the first time around.'''
    text = re.sub('[‘’“”…]', '', text)
    text = re.sub('[\r\n]', '', text)
    return text

round2 = lambda x: clean_text_round2(x)
```

```python
def tokenize(text):
    '''
    Input: Text String (str)

    Process:
    1. Tokenize text into tokens
    2. Remove stop words
    3. Lemmatize

    Output: List of text tokens for string
    '''
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()
    tokens = [w for w in tokens if not w in stop_words]
    tokens = [lemmatizer.lemmatize(w.lower().strip()) for w in tokens]
    return tokens
```

Data Understanding

The Data below was collected from Kaggle.com... it contains 3 csv files namely, `calender.csvlisting.csvreviews.csv`.

```python
# Loading the dataset
df_reviews = pd.read_csv("Data/reviews.csv")
df_cal = pd.read_csv("Data/calendar.csv")
listing = pd.read_csv("Data/listings.csv")
```

```python
# checking out the data types of the columns

df_reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 84849 entries, 0 to 84848
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   listing_id     84849 non-null  int64 
 1   id             84849 non-null  int64 
 2   date           84849 non-null  object
 3   reviewer_id    84849 non-null  int64 
 4   reviewer_name  84849 non-null  object
 5   comments       84831 non-null  object
dtypes: int64(3), object(3)
memory usage: 3.9+ MB
```

```python
# investigating the review dataset's first 5 rows
df_reviews.head()
```

```
   listing_id        id        date  reviewer_id reviewer_name  \
0     7202016  38917982  2015-07-19     28943674        Bianca
1     7202016  39087409  2015-07-20     32440555         Frank
2     7202016  39820030  2015-07-26     37722850           Ian
3     7202016  40813543  2015-08-02     33671805        George
4     7202016  41986501  2015-08-10     34959538          Ming


comments
0
Cute and cozy place. Perfect location to everything!
1  Kelly has a great room in a very central location. \r\nBeautiful
building , architecture and a style that we really like. \r\nWe felt
guite at home here and wish we had spent more time.\r\nWent for a walk
and found Seattle Center with a major food festival in progress. What
a treat.\r\nVisited the Space Needle and the Chihuly Glass exhibit.
Then Pikes Place Market. WOW.  Thanks for a great stay.
2      Very spacious apartment, and in a great neighborhood.  This is
the kind of apartment I wish I had!\r\n\r\nDidn't really get to meet
Kelly until I was on my out, but she was always readily available by
phone. \r\n\r\nI believe the only "issue" (if you want to call it
that) was finding a place to park, but I sincerely doubt its easy to
park anywhere in a residential area after 5 pm on a Friday
3
Close to Seattle Center and all it has to offer - ballet, theater,
museum, Space Needle, restaurants of all ilk just blocks away, and the
Metropolitan (probably the coolest grocer you'll ever find). Easy to
find and Kelly was warm, welcoming, and really interesting to talk to.

4                                                               Kelly
was a great host and very accommodating in a great neighborhood. She
has some great coffee and while I wasn't around much during my stay
the time I spent interacting with her was very pleasant. \r\n\r\nThe
apartment is in a great location and very close to the Seattle Center.
The neighborhood itself has a lot of good food as well!
```

Investigating the shapes

```
# investigating the shape of the dataset
df_reviews.shape
```

```
(84849, 6)
```

```
df_cal.shape
```

```
(1393570, 4)
```

```
listing.shape
```

```
(3818, 92)
```

Missing Data

```
# checking for missing values
missing_values(df_reviews)
```

```
          Total missing values  Percentage(%)
comments                    18       0.021214
```

Duplicated Data

```
# Checking for duplicated data

df_reviews.duplicated().value_counts()
```

```
False    84849
dtype: int64
```

```
listing.duplicated().value_counts()
```

```
False    3818
dtype: int64
```

Summary of Data Understanding

The favorite Data are reviews and calender.

`reviews.csv` the most promising column here is the comments part... such a rich treasure trove of vital data.

`calender.csv` this data set is also promising as it has a column ,price, which can be used as the target variable.

Now to **merge** the two datasets, calender and reviews using the `listing id` as the primary Key. in order to have the Price column found in listing as the Target Variable.

Data Preparation

Dropping un-neccesarry columns

```
df_reviews.drop(columns=["id", "reviewer_id", "reviewer_name",
"date"], inplace=True)
df_reviews.shape
```

```
(84849, 2)
```

```
# Merge the reviews and prices
reviews_prices = df_reviews.copy()
reviews_prices.head()
```

```
   listing_id  \
0     7202016
1     7202016
2     7202016
3     7202016
4     7202016
```

```
comments
0
Cute and cozy place. Perfect location to everything!
1  Kelly has a great room in a very central location. \r\nBeautiful
building , architecture and a style that we really like. \r\nWe felt
guite at home here and wish we had spent more time.\r\nWent for a walk
and found Seattle Center with a major food festival in progress. What
a treat.\r\nVisited the Space Needle and the Chihuly Glass exhibit.
Then Pikes Place Market. WOW.  Thanks for a great stay.
2       Very spacious apartment, and in a great neighborhood.  This is
the kind of apartment I wish I had!\r\n\r\nDidn't really get to meet
Kelly until I was on my out, but she was always readily available by
phone. \r\n\r\nI believe the only "issue" (if you want to call it
that) was finding a place to park, but I sincerely doubt its easy to
park anywhere in a residential area after 5 pm on a Friday
3
Close to Seattle Center and all it has to offer - ballet, theater,
museum, Space Needle, restaurants of all ilk just blocks away, and the
Metropolitan (probably the coolest grocer you'll ever find). Easy to
find and Kelly was warm, welcoming, and really interesting to talk to.

4                                                               Kelly
was a great host and very accommodating in a great neighborhood. She
has some great coffee and while I wasn't around much during my stay
the time I spent interacting with her was very pleasant. \r\n\r\nThe
apartment is in a great location and very close to the Seattle Center.
The neighborhood itself has a lot of good food as well!
```

```
reviews_prices.shape
```

```
(84849, 2)
```

```
missing_values(reviews_prices)
```

```
          Total missing values  Percentage(%)
comments                    18       0.021214
```

```
# Dropping the missing values...
reviews_prices.dropna(axis=0, how='any',inplace=True)
reviews_prices.head()
reviews_prices.shape
```

```
(84831, 2)
```

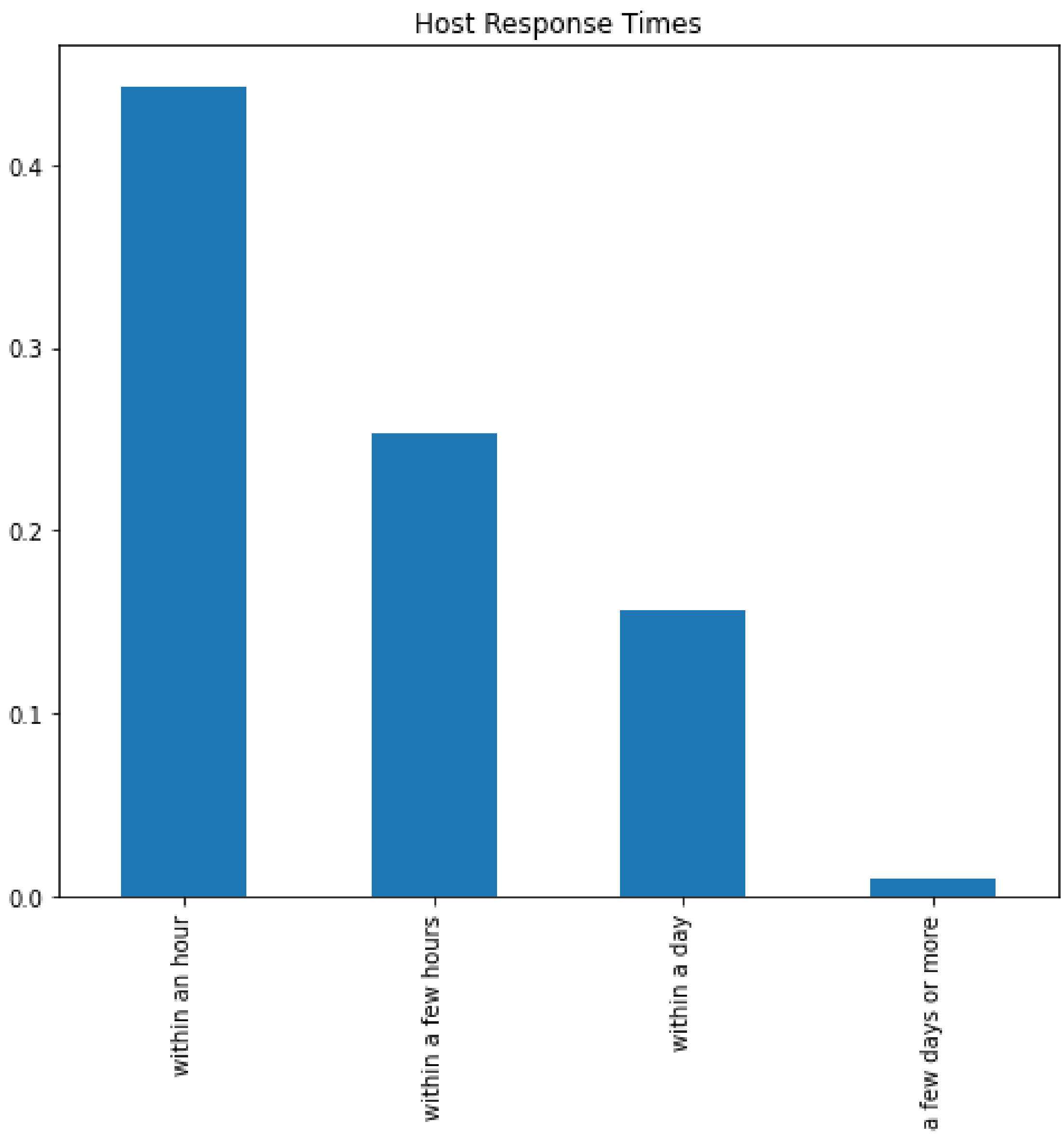
```
# Get the count of reviews grouped by listing_id!
reviews_prices["count"] = reviews_prices.groupby('listing_id',)
['listing_id'].transform("count")

# Checking if price is a string or a numerical data type
reviews_prices.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 84831 entries, 0 to 84848
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   listing_id  84831 non-null  int64
 1   comments    84831 non-null  object
 2   count       84831 non-null  int64
dtypes: int64(2), object(1)
memory usage: 2.6+ MB
```

Price is an object data type instead of an integer type conversion is required.

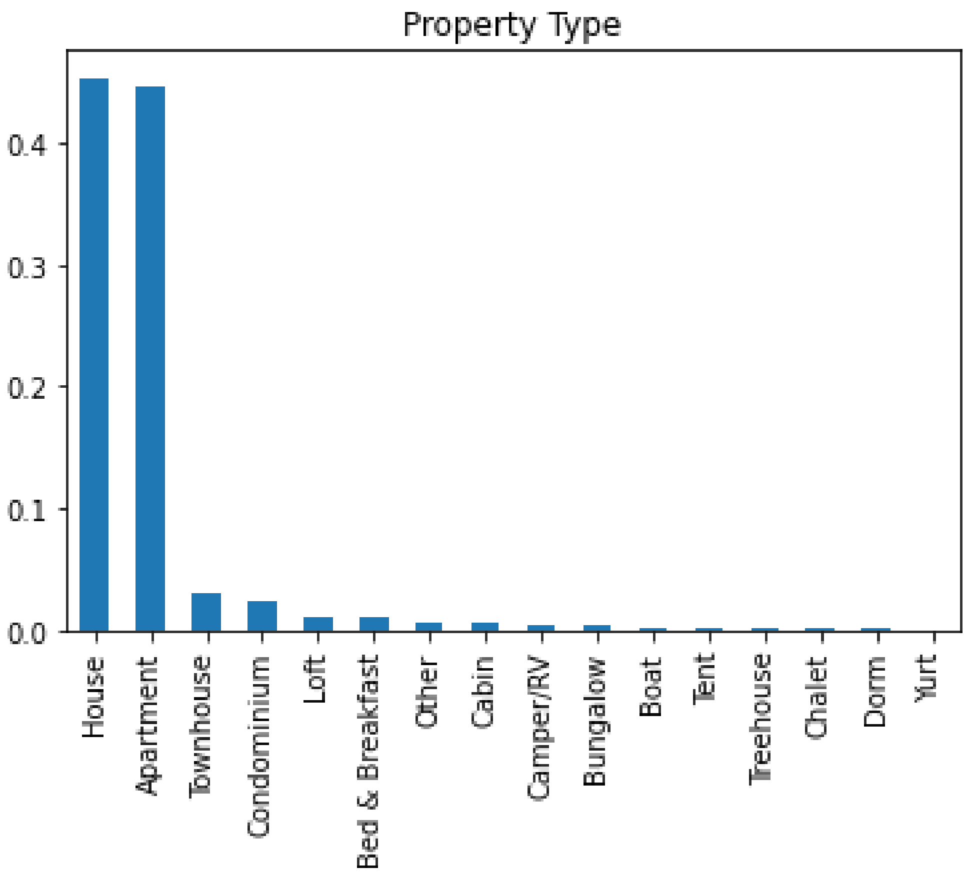
Exploratory Data Analysis

```
# Explore Categorical Feature - host_response_time
fig,ax = plt.subplots(figsize = (8,7))
host_response_vals = listing['host_response_time'].value_counts()
(host_response_vals/listing.shape[0]).plot(kind="bar");
plt.title("Host Response Times");
```



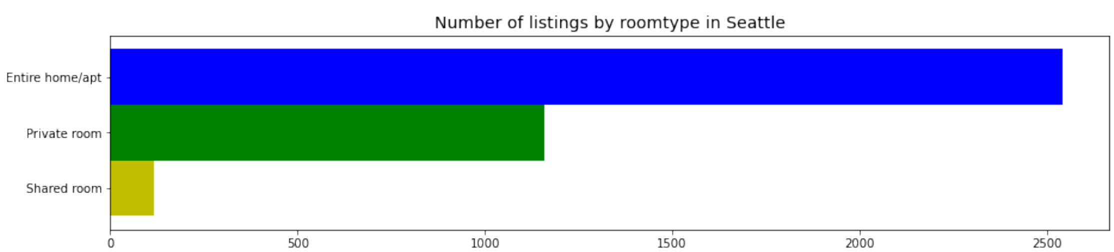
A good percentage of hosts respond to enquireis and complaints within an hour.

```
# Explore Categorical Feature - property_type
prop_vals = listing['property_type'].value_counts()
(prop_vals/listing.shape[0]).plot(kind="bar");
plt.title("Property Type");
```



Most people prefer Airbnb that mainly comprise of houses followed by apartments.

```
#Histogram
freq = listing['room_type'].value_counts().sort_values(ascending=True)
freq.plot.barh(figsize =(15,3), width=1, color=['y','g','b','r'])
plt.title("Number of listings by roomtype in Seattle", fontsize=14)
plt.show();
```

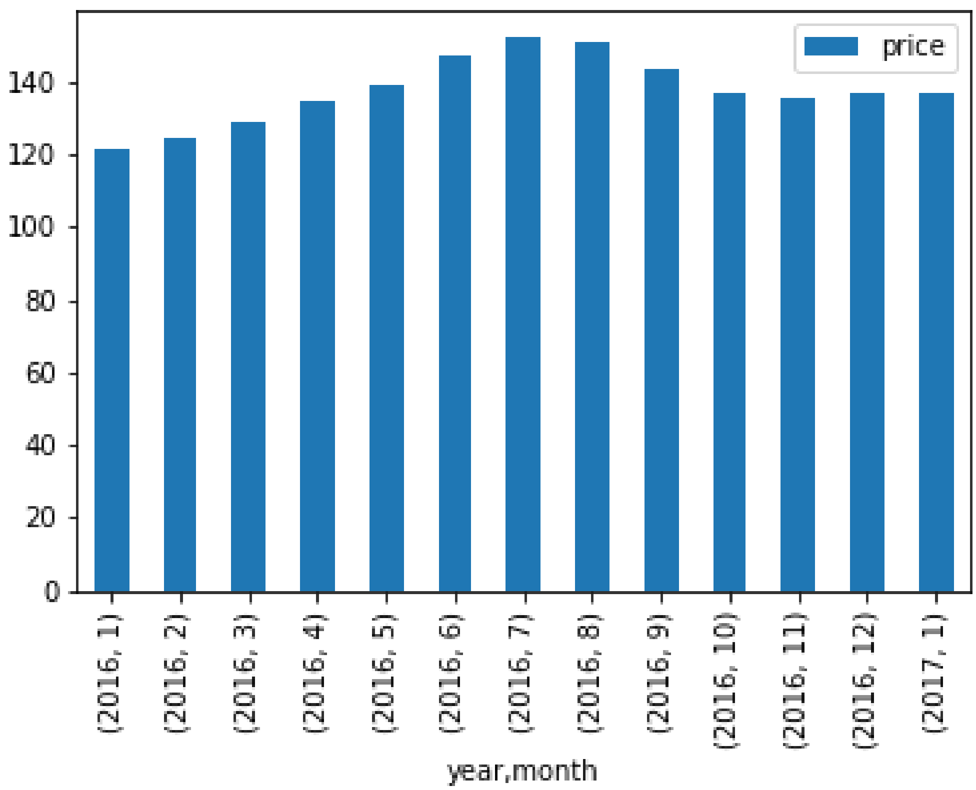


It is evident that people are concerned with their privacy since they prefer having an entire home as an Airbnb compared to a shared or private room.

```
#  clean version of the calender csv file is loaded again
df_cal = pd.read_csv("Data/calendar.csv")
```

clearly people prefer an entire home to a hotel room as an airbnb

```
# Plot the average price by month
df_cal['year'] = pd.DatetimeIndex(df_cal['date']).year
df_cal['month'] = pd.DatetimeIndex(df_cal['date']).month

# In order to average price, we will be converting it to float and
removing the $ sign
df_cal['price'] = df_cal['price'].replace('[\$,]', '',
regex=True).astype(float)

df_cal.groupby(['year','month'])[['price']].mean().plot(kind="bar");
```



Based on the above chart, it shows that June through August are the peak months, with July being the highest. A quick Google search confirms my assumption that these months have the best weather in Seattle with summer in full swing and low chances of rain.

Data Preprocessing

```
sia = SentimentIntensityAnalyzer()
reviews_prices['neg'] = reviews_prices['comments'].apply(lambda
x:sia.polarity_scores(x)['neg'])
reviews_prices['neu'] = reviews_prices['comments'].apply(lambda x:
sia.polarity_scores(x)['neu'])
reviews_prices['pos'] = reviews_prices['comments'].apply(lambda
x:sia.polarity_scores(x)['pos'])
```

```
reviews_prices['compound'] = reviews_prices['comments'].apply(lambda
x:sia.polarity_scores(x)['compound'])

reviews_prices.head()
```

```
   listing_id  \
0     7202016
1     7202016
2     7202016
3     7202016
4     7202016


comments  \
0
Cute and cozy place. Perfect location to everything!
1  Kelly has a great room in a very central location. \r\nBeautiful
building , architecture and a style that we really like. \r\nWe felt
guite at home here and wish we had spent more time.\r\nWent for a walk
and found Seattle Center with a major food festival in progress. What
a treat.\r\nVisited the Space Needle and the Chihuly Glass exhibit.
Then Pikes Place Market. WOW.  Thanks for a great stay.
2      Very spacious apartment, and in a great neighborhood.  This is
the kind of apartment I wish I had!\r\n\r\nDidn't really get to meet
Kelly until I was on my out, but she was always readily available by
phone. \r\n\r\nI believe the only "issue" (if you want to call it
that) was finding a place to park, but I sincerely doubt its easy to
park anywhere in a residential area after 5 pm on a Friday
3
Close to Seattle Center and all it has to offer - ballet, theater,
museum, Space Needle, restaurants of all ilk just blocks away, and the
Metropolitan (probably the coolest grocer you'll ever find). Easy to
find and Kelly was warm, welcoming, and really interesting to talk to.

4                                                               Kelly
was a great host and very accommodating in a great neighborhood. She
has some great coffee and while I wasn't around much during my stay
the time I spent interacting with her was very pleasant. \r\n\r\nThe
apartment is in a great location and very close to the Seattle Center.
The neighborhood itself has a lot of good food as well!

   count    neg    neu    pos  compound
0     16  0.000  0.462  0.538    0.7901
1     16  0.000  0.609  0.391    0.9872
2     16  0.043  0.772  0.185    0.8718
3     16  0.035  0.765  0.200    0.8313
4     16  0.000  0.655  0.345    0.9783
```

a sample of the most negative comment based on the sentimentintensityanalyzer()

```
reviews_prices['comp_score'] = reviews_prices['compound'].apply(lambda
c: 'pos' if c >=0 else 'neg')

reviews_prices.head(2)
```

```
   listing_id  \
0     7202016
1     7202016


comments  \
0
Cute and cozy place. Perfect location to everything!
1  Kelly has a great room in a very central location. \r\nBeautiful
building , architecture and a style that we really like. \r\nWe felt
guite at home here and wish we had spent more time.\r\nWent for a walk
and found Seattle Center with a major food festival in progress. What
a treat.\r\nVisited the Space Needle and the Chihuly Glass exhibit.
Then Pikes Place Market. WOW.  Thanks for a great stay.

   count  neg    neu    pos  compound comp_score
0     16  0.0  0.462  0.538    0.7901        pos
1     16  0.0  0.609  0.391    0.9872        pos
```

```
review_later = reviews_prices[['listing_id', 'comments',
'comp_score']]

reviews_prices['comp_score'] =
reviews_prices['comp_score'].map({'pos':1, 'neg':0})
reviews_prices =
reviews_prices.drop(columns=['count','neg','neu','pos', 'compound'])
reviews_prices.head(3)
```

```
   listing_id  \
0     7202016
1     7202016
2     7202016


comments  \
0
Cute and cozy place. Perfect location to everything!
1  Kelly has a great room in a very central location. \r\nBeautiful
building , architecture and a style that we really like. \r\nWe felt
guite at home here and wish we had spent more time.\r\nWent for a walk
and found Seattle Center with a major food festival in progress. What
a treat.\r\nVisited the Space Needle and the Chihuly Glass exhibit.
Then Pikes Place Market. WOW.  Thanks for a great stay.
2      Very spacious apartment, and in a great neighborhood.  This is
the kind of apartment I wish I had!\r\n\r\nDidn't really get to meet
Kelly until I was on my out, but she was always readily available by
```

```
phone. \r\n\r\nI believe the only "issue" (if you want to call it
that) was finding a place to park, but I sincerely doubt its easy to
park anywhere in a residential area after 5 pm on a Friday

   comp_score
0           1
1           1
2           1
```

Data Cleaning

```
#Top 10 common words in the comments with CountVectorizer()
texts= reviews_prices.comments.tolist()

vec = CountVectorizer().fit(texts)
bag_of_words = vec.transform(texts)
sum_words = bag_of_words.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in
vec.vocabulary_.items()]

cvec_df = pd.DataFrame.from_records(words_freq, columns= ['words',
'counts']).sort_values(by="counts", ascending=False)
cvec_df.head(10)
```

```
   words  counts
1    and  289516
47   the  286230
6     to  175381
70   was  163778
12    in  100343
20    we   97600
63    is   89164
65    of   73384
13  very   72242
34   for   68101
```

from the above top ten it is clear that they are all stop words so that should be taken care of next.

```
# raw corpus
data_clean = pd.DataFrame(reviews_prices.comments.apply(round1))
data_clean
```

```
comments
0
cute and cozy place perfect location to everything
1
kelly has a great room in a very central location \r\nbeautiful
building  architecture and a style that we really like \r\nwe felt
guite at home here and wish we had spent more time\r\nwent for a walk
```

and found seattle center with a major food festival in progress what a treat\r\nvisited the space needle and the chihuly glass exhibit then pikes place market wow thanks for a great stay
2
very spacious apartment and in a great neighborhood this is the kind of apartment i wish i had\r\n\r\ndidnt really get to meet kelly until i was on my out but she was always readily available by phone \r\n\r\ni believe the only issue if you want to call it that was finding a place to park but i sincerely doubt its easy to park anywhere in a residential area after pm on a friday
3
close to seattle center and all it has to offer ballet theater museum space needle restaurants of all ilk just blocks away and the metropolitan probably the coolest grocer youll ever find easy to find and kelly was warm welcoming and really interesting to talk to
4
kelly was a great host and very accommodating in a great neighborhood she has some great coffee and while i wasnt around much during my stay the time i spent interacting with her was very pleasant \r\n\r\nthe apartment is in a great location and very close to the seattle center the neighborhood itself has a lot of good food as well
...
...
84844 the description and pictures of the apartment were exactly what we received moreover the place was very nice and we really enjoyed our stay the location was perfect for being near to the conference center in addition we stayed during a weird stretch of weather sometimes hot sometimes cold and they were very accommodating by providing both extra heating and cooling units checkin was smooth and thorough we appreciated receiving sets of keys since we had adults staying in the same place also we forgot to return one of the parking fobs when we checked out and they let us return it late without an issue all together a very enjoyable experience and we would stay here again
84845
we had an excellent stay it was clean and comfortable and very convenient to the convention center and downtown the beds were comfy the apartment was quiet i would stay there again any time
84846
gran ubicación cerca de todo lo atractivo del centro de seattle el departamento está súper equipado para que tengas todo lo necesario doug fue muy amable y servicial disfrutamos mucho la estancia
84847
very good apartement clean and well sized situated next to the convension center take the back entrance and you will be there in no time in walking distance to most everything downtown and close to good places like the six arms just up the street on the negative side can two of the rooms be some what noisy due laundry in the building
84848
breanne was a great host check in was easy she let me in right on schedule and her place was very comfortable and clean just as

described she even left out some postcards and toiletrieswhich were very much appreciated i loved that there was a trader joes across the street i would definitely stay here again cheers

[84831 rows x 1 columns]

```
data_clean = pd.DataFrame(data_clean.comments.apply(round2))
data_clean
```

comments
0
cute and cozy place perfect location to everything
1
kelly has a great room in a very central location beautiful building architecture and a style that we really like we felt guite at home here and wish we had spent more timewent for a walk and found seattle center with a major food festival in progress what a treatvisited the space needle and the chihuly glass exhibit then pikes place market wow thanks for a great stay
2
very spacious apartment and in a great neighborhood this is the kind of apartment i wish i haddidnt really get to meet kelly until i was on my out but she was always readily available by phone i believe the only issue if you want to call it that was finding a place to park but i sincerely doubt its easy to park anywhere in a residential area after pm on a friday
3
close to seattle center and all it has to offer ballet theater museum space needle restaurants of all ilk just blocks away and the metropolitan probably the coolest grocer youll ever find easy to find and kelly was warm welcoming and really interesting to talk to
4
kelly was a great host and very accommodating in a great neighborhood she has some great coffee and while i wasnt around much during my stay the time i spent interacting with her was very pleasant the apartment is in a great location and very close to the seattle center the neighborhood itself has a lot of good food as well
...
...
84844 the description and pictures of the apartment were exactly what we received moreover the place was very nice and we really enjoyed our stay the location was perfect for being near to the conference center in addition we stayed during a weird stretch of weather sometimes hot sometimes cold and they were very accommodating by providing both extra heating and cooling units checkin was smooth and thorough we appreciated receiving sets of keys since we had adults staying in the same place also we forgot to return one of the parking fobs when we checked out and they let us return it late without an issue all together a very enjoyable experience and we would stay here again

```
84845
we had an excellent stay it was clean and comfortable and very
convenient to the convention center and downtown the beds were comfy
the apartment was quiet i would stay there again any time
84846
gran ubicación cerca de todo lo atractivo del centro de seattle el
departamento está súper equipado para que tengas todo lo necesario
doug fue muy amable y servicial disfrutamos mucho la estancia
84847
very good apartement clean and well sized situated next to the
convension center take the back entrance and you will be there in no
time in walking distance to most everything downtown and close to good
places like the six arms just  up the street on the negative side can
two of the rooms be some what noisy due laundry in the building
84848
breanne was a great host check in was easy she let me in right on
schedule and her place was very comfortable and clean just as
described  she even left out some postcards and toiletrieswhich were
very much appreciated i loved that there was a trader joes across the
street i would definitely stay here again cheers

[84831 rows x 1 columns]
```

```python
data_clean['tokenized'] = data_clean.apply(lambda row:
nltk.word_tokenize(row['comments']), axis=1)

frequent_words = []
for message in data_clean['tokenized']:
    frequent_words.extend([word for word in message if len(word)> 5 ])

wnl = WordNetLemmatizer()
lemmatized =[]
for lemma in frequent_words:
    lemma = wnl.lemmatize(lemma)
    lemmatized.append(lemma)
len(lemmatized)
```

```
1514820
```

Word Cloud and TopGrams

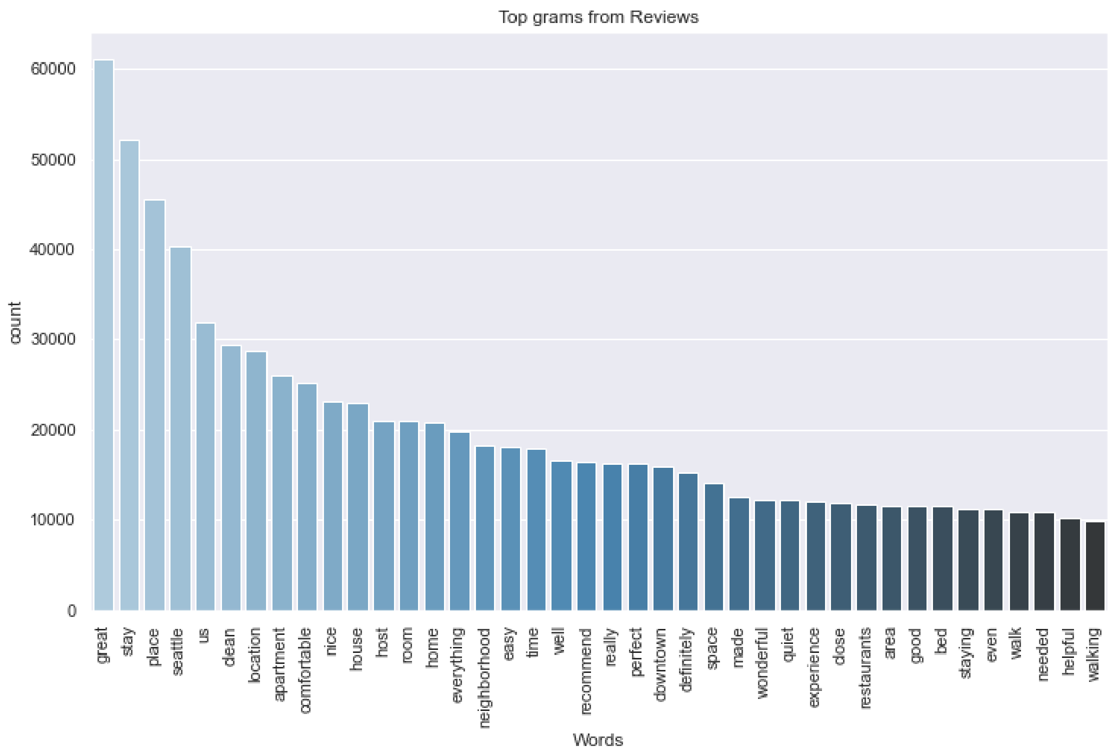
To assess the preferred words guests used to describe their experience, I also pulled out top unigram, top bigrams and top trigrams, as well as created 2 wordclouds for the positive and negative comments.

Step 1: Count the frequency of top 1 word, 2-word phrases, 3-word phrases

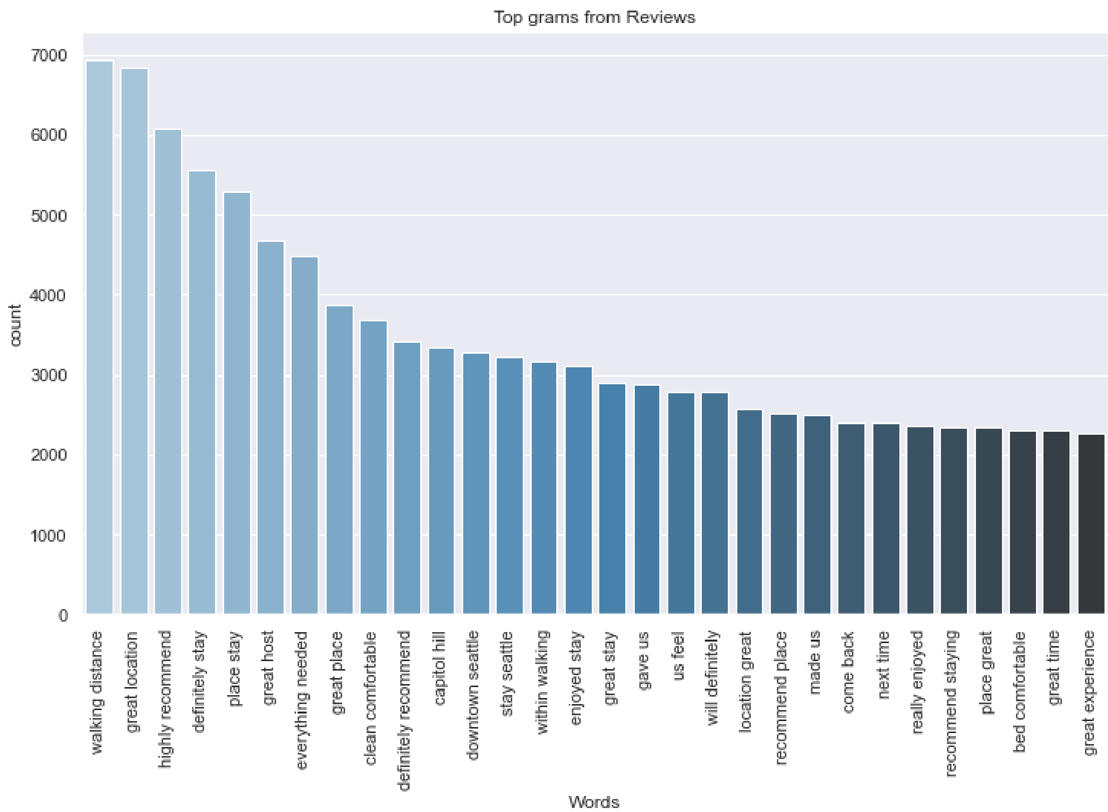
Step 2: Visualize the top n-grams with seaborn package and popular words in comment texts using WordCloud package

```
doc = list(reviews_prices['comments'])
tfidf_vectorizer=TfidfVectorizer(use_idf=True, max_features =
5000,stop_words= STOPWORDS)

# Step 1: count the frequency by grams
def get_top_n_gram(corpus,gram, n):
    """Return n word for unigram, bigram, trigram, etc. from corpus

    Args:
        corpus (str): text to obtain the grams from
        gram (int): number of word in the phrase to extract. For
example, 1: unigram, 2:bigram, 3:trigram
        n (int): number of phrases to be extracted from text. For
example, 30 top unigrams, 20 top bigrams, etc.

    Returns:
        dataframe: a dataframe of top n words and their count in the
text
        """



    vec = CountVectorizer(ngram_range=(gram, gram),stop_words =
STOPWORDS,)
    bag_of_words = vec.fit_transform(doc)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in
vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

uni_grams = pd.DataFrame(get_top_n_gram(data_clean['comments'],1,40),
columns=['Words','count'])
bi_grams = pd.DataFrame(get_top_n_gram(data_clean['comments'],2,30),
columns=['Words','count'])

tri_grams = pd.DataFrame(get_top_n_gram(data_clean['comments'],3,15),
columns=['Words','count'])

# Step 3: Visualize top grams
def plot_gram(data):
    """Visualize the top grams dataframe with seaborn barplot

    Args:
        data: dataframe of top words and their count returned from
get_top_n_gram function

    Returns:
```

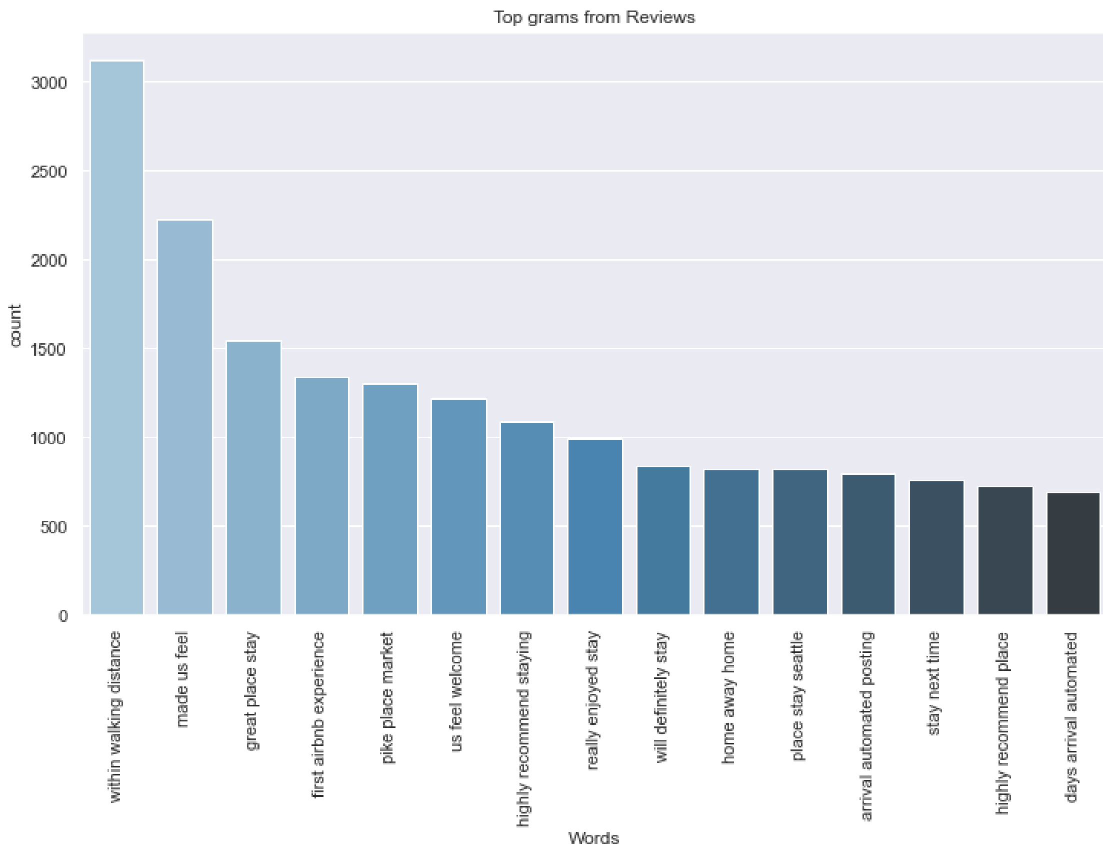
```
        barplot: barplot of top words and their count
    """

    data.sort_values(by=['count'], ascending = False)
    sns.set(rc={'figure.figsize':(12,7)})
    ax = sns.barplot(x='Words', y='count', data = data, palette =
'Blues_d');
    ax.set_xticklabels(labels = data['Words'], rotation=90);
    ax.set_title('Top grams from Reviews');

plot_gram(uni_grams)
```




```
plot_gram(bi_grams)
```




```
plot_gram(tri_grams)
```



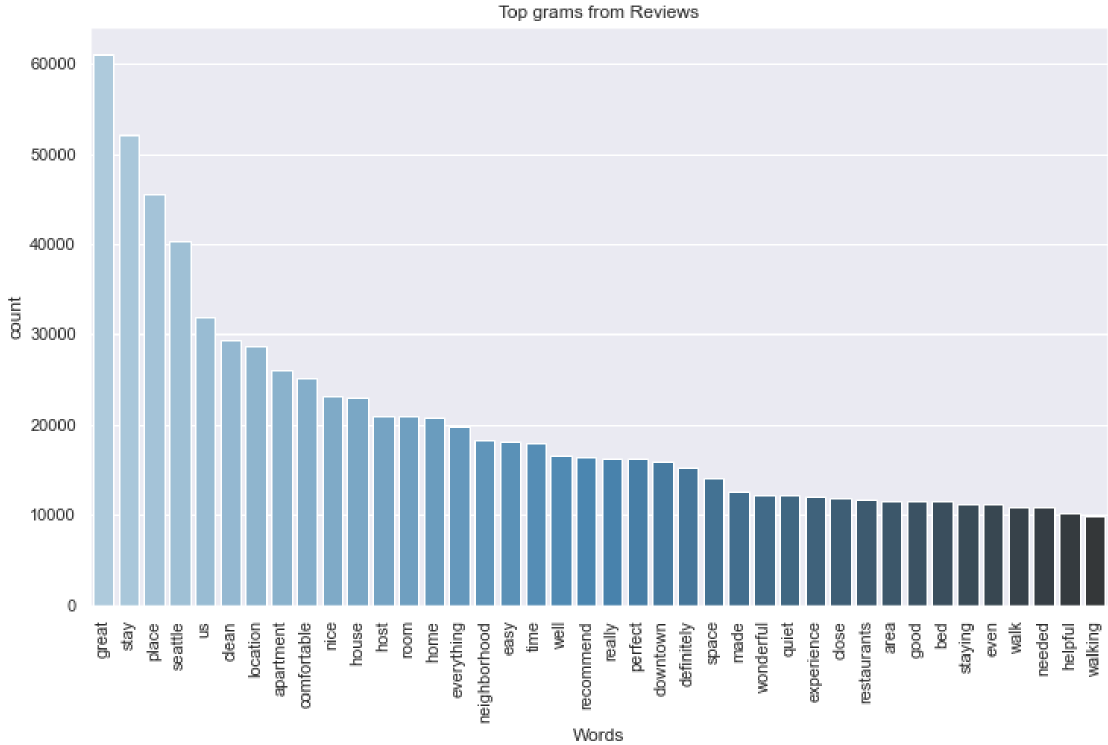

```
data_clean['tokenized'] = data_clean.apply(lambda row:
nltk.word_tokenize(row['comments']), axis=1)

words = []
for message in data_clean['tokenized']:
    words.extend([word for word in message if word not in STOPWORDS])
plt.figure(figsize=(20,12))
wordcloud = WordCloud(width = 3000, height = 1500).generate("
".join(words))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```


```
sentiment_label = review_later['comp_score'].factorize()
sentiment_label
```

```
(array([0, 0, 0, ..., 0, 0, 0]), Index(['pos', 'neg'],
dtype='object'))
```

Below we attempt to isolate all the negative review and repeat the above steps to try and find if there are any negative actions that host could learn to avoid.

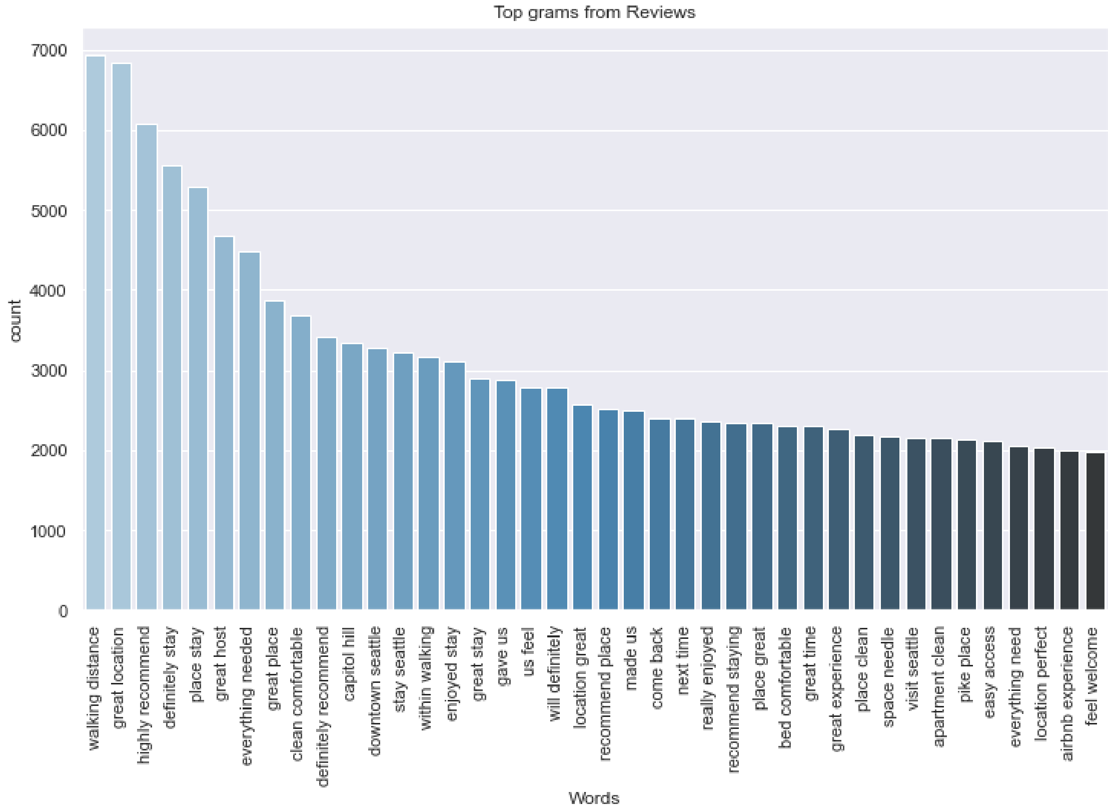
```
# can we look for the common words in negative.
neg_reviews = reviews_prices.loc[reviews_prices['comp_score'] == 0]
unigram_neg =
pd.DataFrame(get_top_n_gram(neg_reviews['comments'],1,40),
columns=['Words','count'])
plot_gram(unigram_neg)
```



from the unigram above above it seems there is no apparent difference.

let us try it on bigrams next

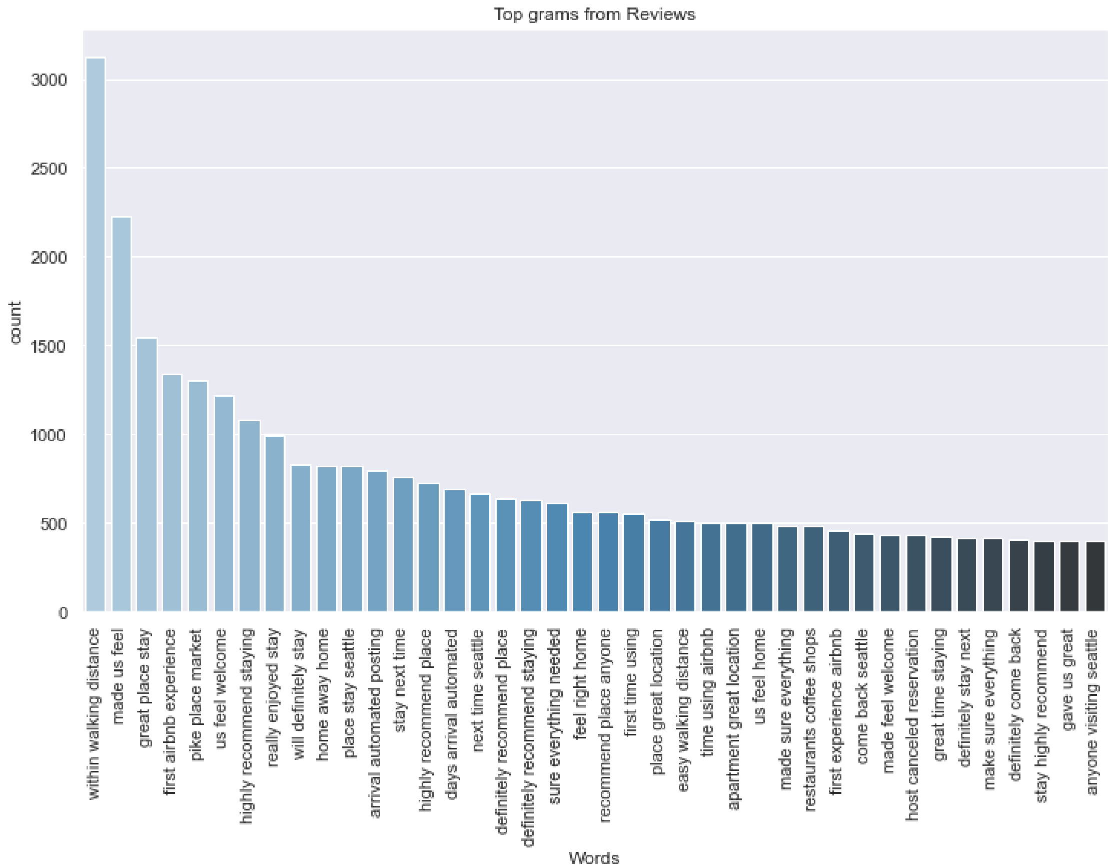
```
bigram_neg =
pd.DataFrame(get_top_n_gram(neg_reviews['comments'],2,40),
columns=['Words','count'])
plot_gram(bigram_neg)
```



Just us above no observable diffefrence s observed.

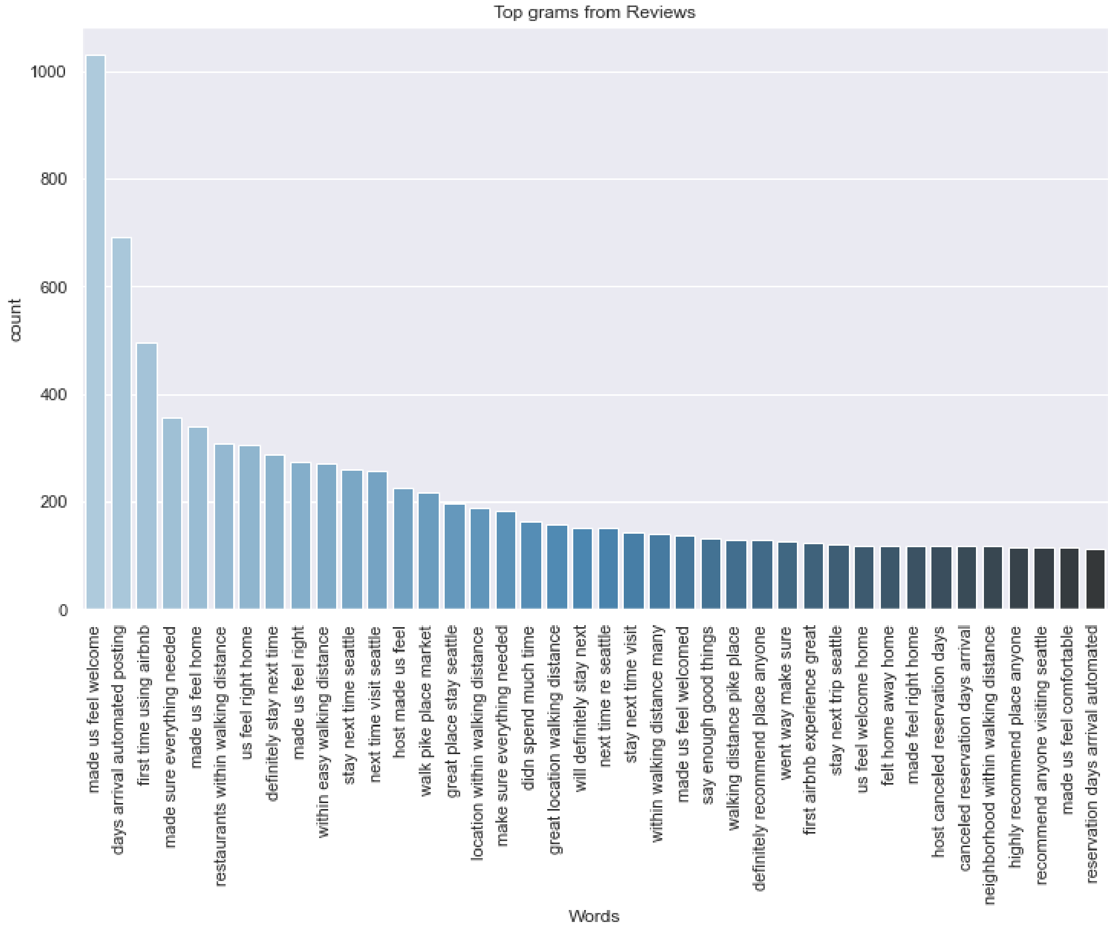
Onto trigrams

```
trigram_neg =
pd.DataFrame(get_top_n_gram(neg_reviews['comments'],3,40),
columns=['Words','count'])
plot_gram(trigram_neg)
```



from here ,in the trigram, we notice the first negative combined words meetings, `host canceled resevartion`.

```
quadgram_neg =
pd.DataFrame(get_top_n_gram(neg_reviews['comments'],4,40),
columns=['Words','count'])
plot_gram(quadgram_neg)
```



Lets try a Quadgram as a Hail Mary. We got two common combinations, `canceled reservation days arrival` and `hast canceled reservation days`

Modelling

```
review = review_later['comments'].values
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(review)
vocab_size = len(tokenizer.word_index) + 1

max_words=5000
max_len = 200

encoded_docs = tokenizer.texts_to_sequences(review)

padded_sequence = pad_sequences(encoded_docs, maxlen=200)

sentiment_label = review_later['comp_score'].factorize()
sentiment_label
```

```
(array([0, 0, 0, ..., 0, 0, 0]), Index(['pos', 'neg'],
dtype='object'))
```

Build the classifier

For sentiment analysis project, we use LSTM layers in the machine learning model. The architecture of our model consists of an embedding layer, an LSTM layer, and a Dense layer at the end. To avoid overfitting, we introduced the Dropout mechanism in-between the LSTM layers.

LSTM stands for Long Short Term Memory Networks. It is a variant of Recurrent Neural Networks. Recurrent Neural Networks are usually used with sequential data such as text and audio. Usually, while computing an embedding matrix, the meaning of every word and its calculations (which are called hidden states) are stored. If the reference of a word, let's say a word is used after 100 words in a text, then all these calculations RNNs cannot store in its memory. That's why RNNs are not capable of learning these long-term dependencies.

Train the sentiment analysis model for 5 epochs on the whole dataset with a batch size of 32 and a validation split of 20%.

Base Model

Adam Optimizer

```
from numpy.random import seed
seed(1)

tf.random.set_seed(42)
embedding_vector_length = 20
model = Sequential()
model.add(Embedding(vocab_size, embedding_vector_length,
input_length=200))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',
metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 200, 20)           878720
_________________________________________________________________
flatten (Flatten)            (None, 4000)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 4001
=================================================================
Total params: 882,721
Trainable params: 882,721
Non-trainable params: 0
_________________________________________________________________
None
```

```
history =
model.fit(padded_sequence,sentiment_label[0],validation_split=0.35,
epochs=10, batch_size=1000)
```

```
Epoch 1/10
56/56 [==============================] - 2s 28ms/step - loss: 0.1860 -
accuracy: 0.9825 - val_loss: 0.0562 - val_accuracy: 0.9898
Epoch 2/10
56/56 [==============================] - 1s 16ms/step - loss: 0.0509 -
accuracy: 0.9903 - val_loss: 0.0508 - val_accuracy: 0.9898
Epoch 3/10
56/56 [==============================] - 1s 16ms/step - loss: 0.0452 -
accuracy: 0.9905 - val_loss: 0.0459 - val_accuracy: 0.9901
Epoch 4/10
56/56 [==============================] - 1s 15ms/step - loss: 0.0391 -
accuracy: 0.9910 - val_loss: 0.0413 - val_accuracy: 0.9909
Epoch 5/10
56/56 [==============================] - 1s 15ms/step - loss: 0.0341 -
accuracy: 0.9917 - val_loss: 0.0377 - val_accuracy: 0.9910
Epoch 6/10
56/56 [==============================] - 1s 15ms/step - loss: 0.0298 -
accuracy: 0.9920 - val_loss: 0.0348 - val_accuracy: 0.9910
Epoch 7/10
56/56 [==============================] - 1s 18ms/step - loss: 0.0261 -
accuracy: 0.9926 - val_loss: 0.0325 - val_accuracy: 0.9914
Epoch 8/10
56/56 [==============================] - 1s 23ms/step - loss: 0.0227 -
accuracy: 0.9932 - val_loss: 0.0307 - val_accuracy: 0.9915
Epoch 9/10
56/56 [==============================] - 1s 22ms/step - loss: 0.0198 -
accuracy: 0.9939 - val_loss: 0.0297 - val_accuracy: 0.9916
Epoch 10/10
56/56 [==============================] - 1s 23ms/step - loss: 0.0173 -
accuracy: 0.9944 - val_loss: 0.0287 - val_accuracy: 0.9918
```

```
embedding_vector_length =20
model1 = Sequential()
model1.add(Embedding(vocab_size, embedding_vector_length,
input_length=200))
model1.add(Flatten())
model1.add(Dense(10, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))
model1.compile(loss='binary_crossentropy',optimizer='adam',
metrics=['accuracy'])
print(model1.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 200, 20)           878720
```

```
_________________________________________________________________
flatten_1 (Flatten)          (None, 4000)              0

_________________________________________________________________
dense_1 (Dense)              (None, 10)                40010

_________________________________________________________________
dense_2 (Dense)              (None, 1)                 11
=================================================================
Total params: 918,741
Trainable params: 918,741
Non-trainable params: 0

_________________________________________________________________
None
```

```
history =
model1.fit(padded_sequence,sentiment_label[0],validation_split=0.4,
epochs=10, batch_size=1000)
```

```
Epoch 1/10
51/51 [==============================] - 2s 30ms/step - loss: 0.1493 -
accuracy: 0.9790 - val_loss: 0.0603 - val_accuracy: 0.9894
Epoch 2/10
51/51 [==============================] - 1s 28ms/step - loss: 0.0490 -
accuracy: 0.9906 - val_loss: 0.0502 - val_accuracy: 0.9894
Epoch 3/10
51/51 [==============================] - 1s 25ms/step - loss: 0.0409 -
accuracy: 0.9908 - val_loss: 0.0443 - val_accuracy: 0.9901
Epoch 4/10
51/51 [==============================] - 1s 23ms/step - loss: 0.0341 -
accuracy: 0.9919 - val_loss: 0.0393 - val_accuracy: 0.9906
Epoch 5/10
51/51 [==============================] - 1s 23ms/step - loss: 0.0282 -
accuracy: 0.9925 - val_loss: 0.0344 - val_accuracy: 0.9911
Epoch 6/10
51/51 [==============================] - 1s 22ms/step - loss: 0.0218 -
accuracy: 0.9935 - val_loss: 0.0309 - val_accuracy: 0.9913
Epoch 7/10
51/51 [==============================] - 1s 23ms/step - loss: 0.0165 -
accuracy: 0.9947 - val_loss: 0.0299 - val_accuracy: 0.9914
Epoch 8/10
51/51 [==============================] - 1s 22ms/step - loss: 0.0124 -
accuracy: 0.9960 - val_loss: 0.0295 - val_accuracy: 0.9912
Epoch 9/10
51/51 [==============================] - 1s 22ms/step - loss: 0.0091 -
accuracy: 0.9971 - val_loss: 0.0294 - val_accuracy: 0.9914
Epoch 10/10
51/51 [==============================] - 1s 25ms/step - loss: 0.0064 -
accuracy: 0.9982 - val_loss: 0.0307 - val_accuracy: 0.9913
```

```
embedding_vector_length =20
model2 = Sequential()
model2.add(Embedding(vocab_size, embedding_vector_length,
```

```
input_length=200))
model2.add(Flatten())
model2.add(Dense(30, activation='relu'))
model2.add(Dropout(0.2))
model2.add(Dense(12, activation='relu'))
model2.add(Dropout(0.3))
model2.add(Dense(1, activation='sigmoid'))
model2.compile(loss='binary_crossentropy',optimizer='adam',
metrics=['accuracy'])
print(model2.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 200, 20)           878720
_________________________________________________________________
flatten_2 (Flatten)          (None, 4000)              0
_________________________________________________________________
dense_3 (Dense)              (None, 30)                120030
_________________________________________________________________
dropout (Dropout)            (None, 30)                0
_________________________________________________________________
dense_4 (Dense)              (None, 12)                372
_________________________________________________________________
dropout_1 (Dropout)          (None, 12)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 13
=================================================================
Total params: 999,135
Trainable params: 999,135
Non-trainable params: 0
_________________________________________________________________
None
```

```
history =
model2.fit(padded_sequence,sentiment_label[0],validation_split=0.2,
epochs=5, batch_size=1000)
```

```
Epoch 1/5
68/68 [==============================] - 2s 33ms/step - loss: 0.1773 -
accuracy: 0.9622 - val_loss: 0.0592 - val_accuracy: 0.9902
Epoch 2/5
68/68 [==============================] - 2s 33ms/step - loss: 0.0675 -
accuracy: 0.9895 - val_loss: 0.0453 - val_accuracy: 0.9903
Epoch 3/5
68/68 [==============================] - 3s 39ms/step - loss: 0.0454 -
accuracy: 0.9903 - val_loss: 0.0355 - val_accuracy: 0.9907
Epoch 4/5
68/68 [==============================] - 2s 30ms/step - loss: 0.0305 -
accuracy: 0.9920 - val_loss: 0.0323 - val_accuracy: 0.9911
```

```
Epoch 5/5
68/68 [==============================] - 2s 30ms/step - loss: 0.0225 -
accuracy: 0.9936 - val_loss: 0.0318 - val_accuracy: 0.9913
```

```
embedding_vector_length =20
model3 = Sequential()
model3.add(Embedding(vocab_size, embedding_vector_length,
input_length=200))
model3.add(Dropout(0.2))
model3.add(Dense(1, activation='sigmoid'))
model3.compile(loss='binary_crossentropy',optimizer='adam',
metrics=['accuracy'])
print(model3.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 200, 20)           878720
_________________________________________________________________
dropout_2 (Dropout)          (None, 200, 20)           0
_________________________________________________________________
dense_6 (Dense)              (None, 200, 1)            21
=================================================================
Total params: 878,741
Trainable params: 878,741
Non-trainable params: 0
_________________________________________________________________
None
```

```
history =
model3.fit(padded_sequence,sentiment_label[0],validation_split=0.4,
epochs=12, batch_size=1000)
```

```
Epoch 1/12
51/51 [==============================] - 4s 71ms/step - loss: 0.6277 -
accuracy: 0.9266 - val_loss: 0.5543 - val_accuracy: 0.9894
Epoch 2/12
51/51 [==============================] - 3s 65ms/step - loss: 0.4776 -
accuracy: 0.9907 - val_loss: 0.3969 - val_accuracy: 0.9896
Epoch 3/12
51/51 [==============================] - 3s 67ms/step - loss: 0.3251 -
accuracy: 0.9907 - val_loss: 0.2581 - val_accuracy: 0.9896
Epoch 4/12
51/51 [==============================] - 4s 71ms/step - loss: 0.2098 -
accuracy: 0.9908 - val_loss: 0.1683 - val_accuracy: 0.9896
Epoch 5/12
51/51 [==============================] - 3s 63ms/step - loss: 0.1410 -
accuracy: 0.9908 - val_loss: 0.1189 - val_accuracy: 0.9897
Epoch 6/12
51/51 [==============================] - 4s 70ms/step - loss: 0.1037 -
```

```
accuracy: 0.9908 - val_loss: 0.0926 - val_accuracy: 0.9897
Epoch 7/12
51/51 [==============================] - 4s 69ms/step - loss: 0.0833 -
accuracy: 0.9908 - val_loss: 0.0781 - val_accuracy: 0.9897
Epoch 8/12
51/51 [==============================] - 3s 64ms/step - loss: 0.0716 -
accuracy: 0.9908 - val_loss: 0.0698 - val_accuracy: 0.9896
Epoch 9/12
51/51 [==============================] - 4s 71ms/step - loss: 0.0648 -
accuracy: 0.9908 - val_loss: 0.0649 - val_accuracy: 0.9896
Epoch 10/12
51/51 [==============================] - 3s 60ms/step - loss: 0.0605 -
accuracy: 0.9908 - val_loss: 0.0620 - val_accuracy: 0.9896
Epoch 11/12
51/51 [==============================] - 3s 62ms/step - loss: 0.0578 -
accuracy: 0.9908 - val_loss: 0.0601 - val_accuracy: 0.9896
Epoch 12/12
51/51 [==============================] - 3s 61ms/step - loss: 0.0560 -
accuracy: 0.9908 - val_loss: 0.0589 - val_accuracy: 0.9896
```

```
embedding_vector_length =20
model4 = Sequential()
model4.add(Embedding(vocab_size, embedding_vector_length,
input_length=200))
model4.add(Dropout(0.2))
model4.add(Dense(1, activation='sigmoid'))
model4.compile(loss='hinge',optimizer='adam', metrics=['accuracy'])
print(model4.summary())
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 200, 20)           878720
_________________________________________________________________
dropout_3 (Dropout)          (None, 200, 20)           0
_________________________________________________________________
dense_7 (Dense)              (None, 200, 1)            21
=================================================================
Total params: 878,741
Trainable params: 878,741
Non-trainable params: 0
_________________________________________________________________
None
```

```
history =
model4.fit(padded_sequence,sentiment_label[0],validation_split=0.4,
epochs=12, batch_size=1000)
```

```
Epoch 1/12
51/51 [==============================] - 4s 71ms/step - loss: 1.4581 -
```

```
accuracy: 0.8940 - val_loss: 1.4142 - val_accuracy: 0.9895
Epoch 2/12
51/51 [==============================] - 3s 66ms/step - loss: 1.3635 -
accuracy: 0.9907 - val_loss: 1.3041 - val_accuracy: 0.9896
Epoch 3/12
51/51 [==============================] - 3s 68ms/step - loss: 1.2471 -
accuracy: 0.9907 - val_loss: 1.1893 - val_accuracy: 0.9896
Epoch 4/12
51/51 [==============================] - 4s 79ms/step - loss: 1.1481 -
accuracy: 0.9908 - val_loss: 1.1094 - val_accuracy: 0.9897
Epoch 5/12
51/51 [==============================] - 4s 71ms/step - loss: 1.0873 -
accuracy: 0.9908 - val_loss: 1.0655 - val_accuracy: 0.9897
Epoch 6/12
51/51 [==============================] - 4s 76ms/step - loss: 1.0545 -
accuracy: 0.9908 - val_loss: 1.0422 - val_accuracy: 0.9897
Epoch 7/12
51/51 [==============================] - 4s 77ms/step - loss: 1.0366 -
accuracy: 0.9908 - val_loss: 1.0290 - val_accuracy: 0.9897
Epoch 8/12
51/51 [==============================] - 4s 70ms/step - loss: 1.0261 -
accuracy: 0.9908 - val_loss: 1.0210 - val_accuracy: 0.9897
Epoch 9/12
51/51 [==============================] - 4s 71ms/step - loss: 1.0195 -
accuracy: 0.9908 - val_loss: 1.0159 - val_accuracy: 0.9897
Epoch 10/12
51/51 [==============================] - 4s 72ms/step - loss: 1.0151 -
accuracy: 0.9908 - val_loss: 1.0124 - val_accuracy: 0.9897
Epoch 11/12
51/51 [==============================] - 4s 70ms/step - loss: 1.0120 -
accuracy: 0.9908 - val_loss: 1.0100 - val_accuracy: 0.9897
Epoch 12/12
51/51 [==============================] - 4s 79ms/step - loss: 1.0098 -
accuracy: 0.9908 - val_loss: 1.0082 - val_accuracy: 0.9897
```

Naive Bayes

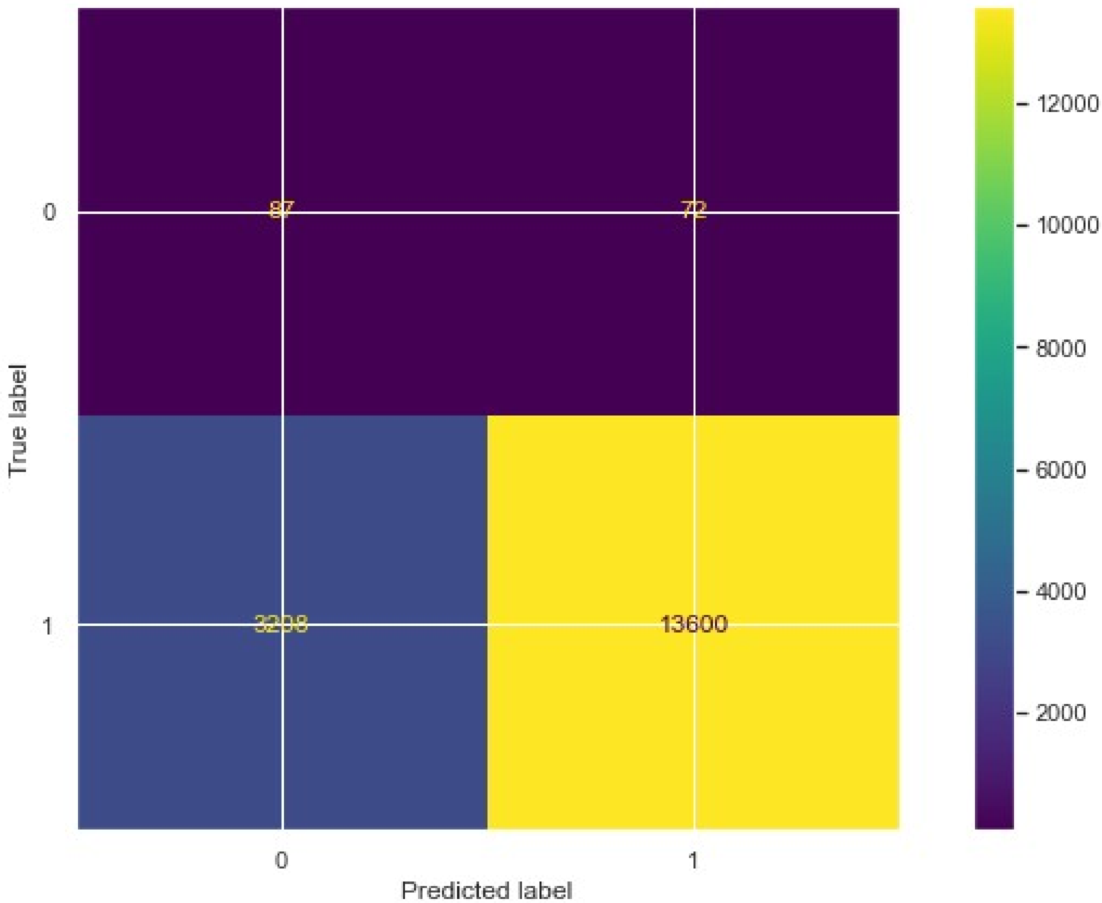
```
doc = list(reviews_prices['comments'])
tfidf_vectorizer=TfidfVectorizer(use_idf=True, max_features = 5000)

tfidf_vectorizer_vectors=tfidf_vectorizer.fit_transform(doc)

Params_tune = {'var_smoothing':[9e-5,7e-5,5e-5,9e-4]}

doc = list(reviews_prices['comments'])
tfidf_vectorizer=TfidfVectorizer(use_idf=True, max_features =
5000,stop_words= STOPWORDS)

tfidf_vectorizer_vectors=tfidf_vectorizer.fit_transform(doc)

X = tfidf_vectorizer_vectors.toarray()
```

```
y = reviews_prices['comp_score']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2)

gnb = GaussianNB()

grid = GridSearchCV(gnb,param_grid=Params_tune,cv = 4,
scoring='accuracy')
model_grid = grid.fit(X_train, y_train)


print(model_grid.best_params_)
print(model_grid.best_score_)
```

```
{'var_smoothing': 0.0009}
0.8298803489331604
```

```
gnb = GaussianNB(var_smoothing = 9e-04)
gnb.fit(X_train, y_train)
y_pred_train = gnb.predict(X_train)
y_pred_test = gnb.predict(X_test)


print("Training Accuracy score:
"+str(round(accuracy_score(y_train,gnb.predict(X_train)),4)))
print("Testing Accuracy score:
"+str(round(accuracy_score(y_test,gnb.predict(X_test)),4)))
print(classification_report(y_test, gnb.predict(X_test)))
plot_confusion_matrix(gnb,X_test,y_test);
```

```
Training Accuracy score: 0.8107
Testing Accuracy score: 0.8067
              precision    recall  f1-score   support

           0       0.03      0.55      0.05       159
           1       0.99      0.81      0.89     16808

    accuracy                           0.81     16967
   macro avg       0.51      0.68      0.47     16967
weighted avg       0.99      0.81      0.88     16967
```



XGBoost

```
from xgboost import XGBClassifier

# Code here to inspect the values of y_train and y_test
y_train.value_counts().sort_index()

0      684
1    67180
Name: comp_score, dtype: int64
```

The data is okay no need to encode it.

```
# Grid search parameters for hyper tuning
param_grid = {
    'learning_rate':[0.2],
    'colsample_bytree':[0.5],
    'colsample_bylevel':[0.5],
    'colsample_bynode':[0.5],
    'gamma':[0.8],
    'max_depth':[6]
}
# Instantiate XGBClassifier
clf = XGBClassifier()
```

```
grid_clf = GridSearchCV(estimator= clf,param_grid=
param_grid,scoring='accuracy',cv= 3,n_jobs=1)
grid_clf.fit(X_train, y_train)

best_parameters = grid_clf.best_params_

print('Grid Search found the following optimal parameters: ')
for param_name in sorted(best_parameters.keys()):
    print('%s: %r' % (param_name, best_parameters[param_name]))

training_preds = grid_clf.predict(X_train)
test_preds = grid_clf.predict(X_test)
training_accuracy = accuracy_score(y_true=
y_train,y_pred=training_preds)
test_accuracy = accuracy_score(y_true = y_test,y_pred= test_preds)

print('')
print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
print('Validation Accuracy: {:.4}%'.format(test_accuracy * 100))
```

```
Grid Search found the following optimal parameters:
colsample_bylevel: 0.5
colsample_bynode: 0.5
colsample_bytree: 0.5
gamma: 0.8
learning_rate: 0.2
max_depth: 6

Training Accuracy: 99.48%
Validation Accuracy: 99.2%
```

```
results = pd.DataFrame({'Machine Model':
['TensorFlow_Hinge','TensorFlow_binary_crossentropy','NaiveBayes','XGB
oost'],
             'Training Accuracy %':[99.08,99.08,81.07,99.48],
             'Testing Accuracy %':[98.97,98.96,80.67,99.2]})
results
```

```
                   Machine Model  Training Accuracy %  Testing
Accuracy %
0               TensorFlow_Hinge                99.08
98.97
1  TensorFlow_binary_crossentropy                99.08
98.96
2                     NaiveBayes                81.07
80.67
3                        XGBoost                99.48
99.20
```

We have successfully developed python sentiment analysis model. In this machine learning project, we built a binary text classifier that classifies the sentiment of the tweets into positive and negative. We obtained 99% accuracy on validation.

FINDINGS

Most customers from Seattle have a great experience during the stay as most reviews/frequent words were positive.

The results of our study show that reviews are influenced by:

location

neighbourhood

host responsiveness to enquiries

host friendliness

distance of the property from areas such as restaurants

check in process(automation and process length)

cleanliness

comfort

provision of 'everything needed'

public transportation

A good percentage of hosts respond to enquiries and complaints within an hour.

CONCLUSIONS

Beyond price, there are many other factors people consider when booking accommodations. For instance, location and amenities are other practical considerations that attract customers to an Airbnb. Most importantly, online reviews that have consistently grown in importance over the years also determine the rate at which an Airbnb gets booked. The occupancy rate in Seattle tends to be higher during summer where super hosts tend to rank higher compared to regular hosts. To utilize the information gathered from reviews, an appropriate method was selected to analyze the words used within the reviews to parse any data useful for better understanding user behavior, as well as, past and future experiences. Additionally, natural language processing techniques were applied to interpret user review comments associated with the listings. This method of analysis highlighted the text of considerable importance as well as attributed a measure of sentiment which is a dynamic element that provided meaning to the text in addition to significance. The significant elements identified in our model provided justification in selection of which listing characteristics to highlight in our new campaign efforts to increase the reach of Airbnb promotions and capture a wider audience.

Recommendations

Guests value the location and accessibily of their Airbnb listings. Hosts can therefore in their listings show case there unique accessibility to amenities such restaurants, towns, public transportation from their properties to capitalise on the airbnb users need for convinience.

Host friendliness was a reccurent theme in the reviews.One of the best ways for hosts to boost their reviews is by delighting guests with a few extra amenities and being kind.

Since the phrase 'Everything needed' was repeatedly used in customer reviews, Hosts should regularly update their amenities in order to ensure guests have all if not most of what they need for their stay to ensure their comfort.

Hosts should ensure cleaniness of the property especially during check in.This can be achieved by having the property cleaned after every checkout or immediately before any guests checks in and also ensure the house is structurally sound with no visible signs of wear and tear.

It would also go along way to introduce a 1 to 5 rating system to help facilitate the machine learning process in a supervised way in order to avoid the pitfalls of semi-supervised work and also to provide a way for the machine to distinguish the extreme negatives.

To ensure that non-english reviews are also translated before sentiment analysis.