Outline of the notebook

In this notebook, I will show my codes and workings for Natural Language Processing (NLP) techniques used to study the Airbnb reviews. The dataset contains more than 100,000 reviews left by guests who had stayed in Singapore listings in the past.

The models used include:

```
- Bag of Words
- Sentiment Analysis
- Topic Modelling
```

The libraries used include:

```
- LangDetect
- CountVectorizer
- Textblob
- Vader
- Gensim
```

Import basic packages that will be used

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
plt.style.use('fivethirtyeight')
%config InlineBackend.figure_format = 'retina'
```

Load the datasets to be used

In [2]:
```
sglisting = pd.read_csv('./SG listings (1).csv')
```

In [2]:
```
sgreviews = pd.read_csv('./SG reviews.csv')
```

In [291]:
```
sgreviews.shape
```

Out[291]:
```
(101009, 6)
```

Typesetting math: 0%

In [66]:
```
sgreviews.head()
```

Out[66]:

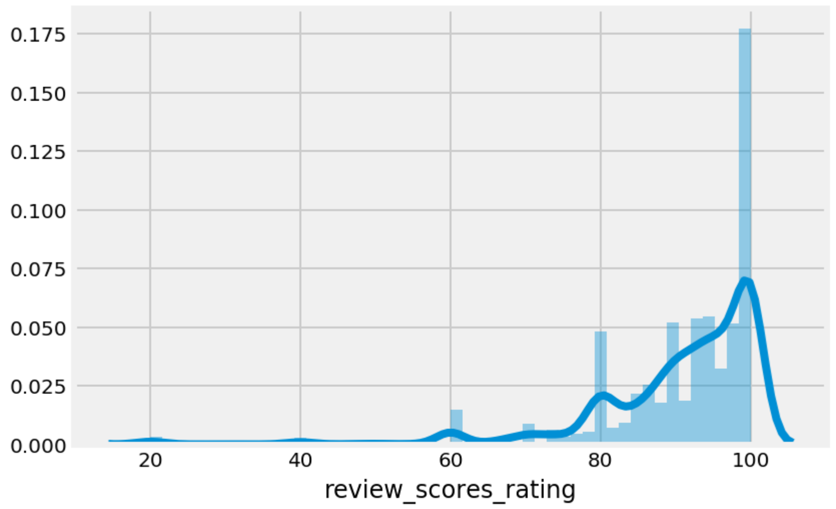
	listing_id	id	date	reviewer_id	reviewer_name	comments	all_cleaned	
0	49091	8243238	2013-10-21	8557223	Jared	Fran was absolutely gracious and welcoming. Ma...	Fran was absolutely gracious and welcoming. Ma...	Franwasabsc
1	50646	11909864	2014-04-18	1356099	James	A comfortable room in a smart condo developmen...	A comfortable room in a smart condo developmen...	Acomfortabl
2	50646	13823948	2014-06-05	15222393	Welli	Stayed over at Sujatha's house for 3 good nigh...	Stayed over at Sujatha's house for 3 good nigh...	Stayedover
3	50646	15117222	2014-07-02	5543172	Cyril	It's been a lovely stay at Sujatha's. The room...	It's been a lovely stay at Sujatha's. The room...	Itsbeena
4	50646	15426462	2014-07-08	817532	Jake	We had a great experience. A nice place, an am...	We had a great experience. A nice place, an am...	Wehadagreat

Visualizing and studying review scores

Review ratings are provided in the listings dataset.

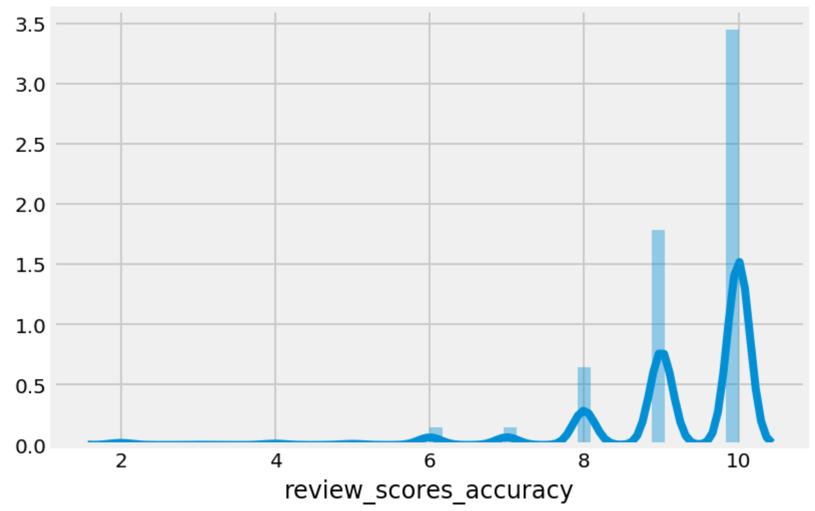
In [80]:
```
# Plot overall reviews scores

sns.distplot(sglisting['review_scores_rating'].dropna())
```

Out[80]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1f5ad6d8>
```

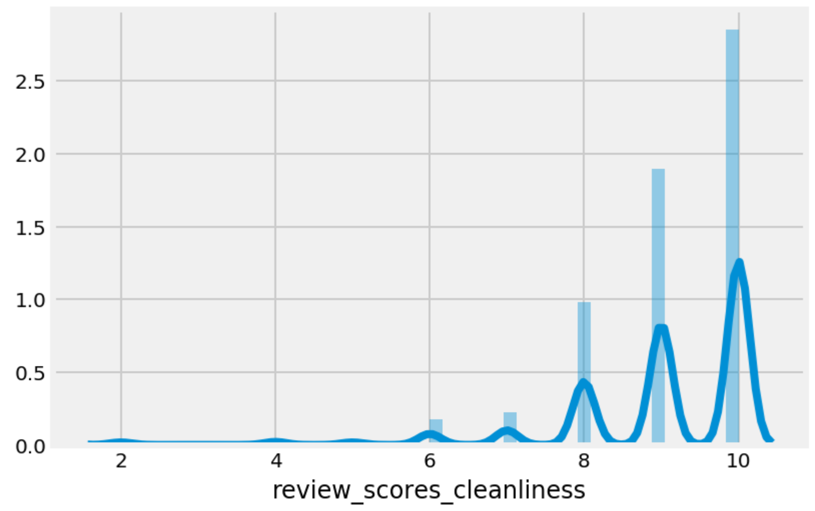


Typesetting math: 0%

```
In [81]: # Plot reviews scores

         sns.distplot(sglisting['review_scores_accuracy'].dropna())

Out[81]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1f5ec940>
```



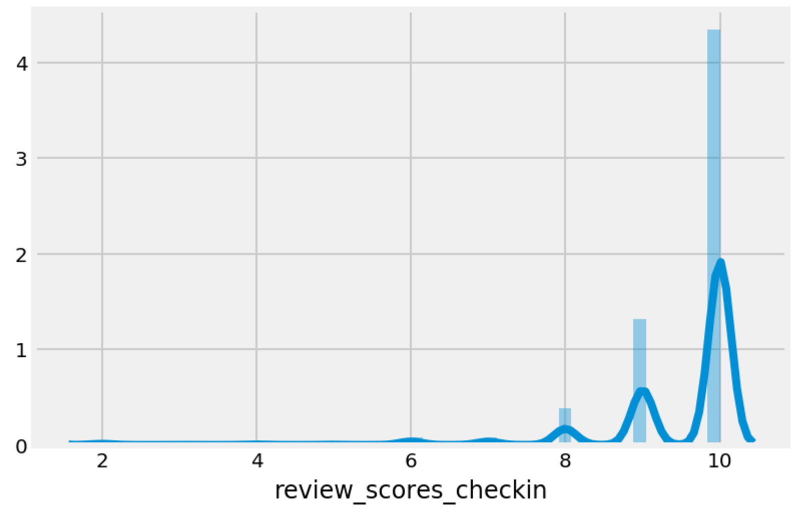
```
In [82]: # Plot overal reviews scores

         sns.distplot(sglisting['review_scores_cleanliness'].dropna())

Out[82]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1f62d978>
```



Typesetting math: 0%

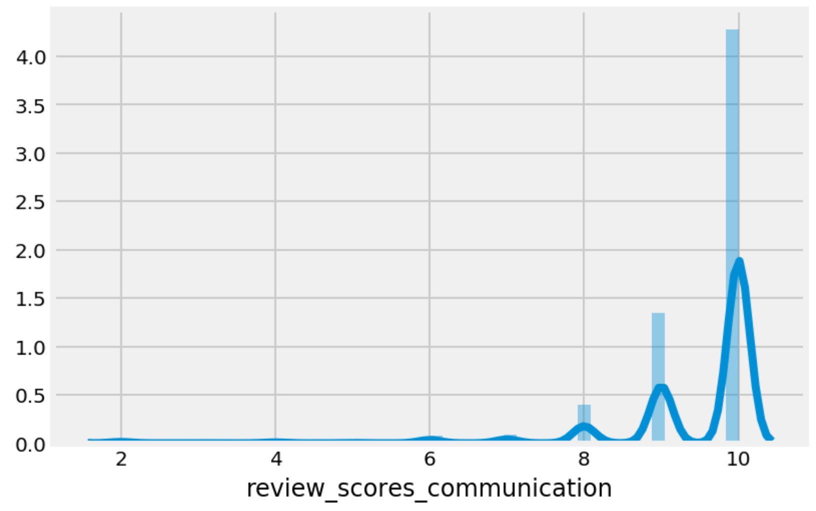
In [83]:
```
# Plot overal reviews scores

sns.distplot(sglisting['review_scores_checkin'].dropna())
```

Out[83]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2017f438>
```



In [84]:
```
# Plot overal reviews scores

sns.distplot(sglisting['review_scores_communication'].dropna())
```

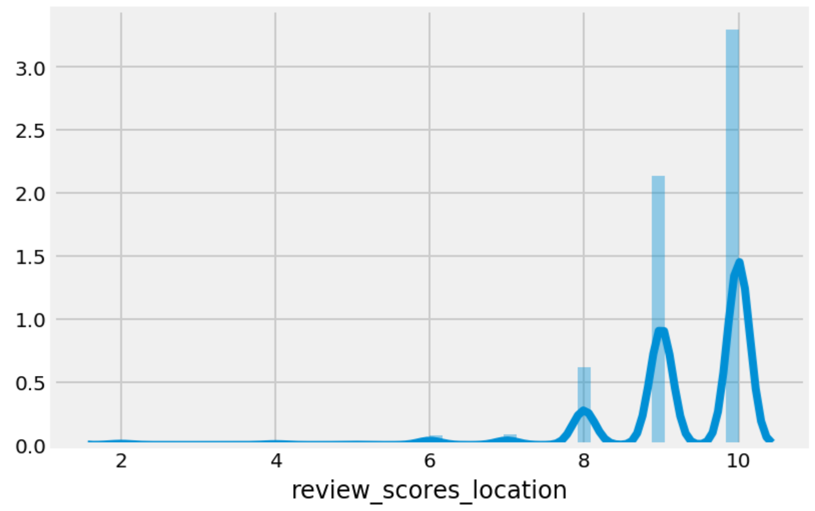
Out[84]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a18ab2518>
```



Typesetting math: 0%

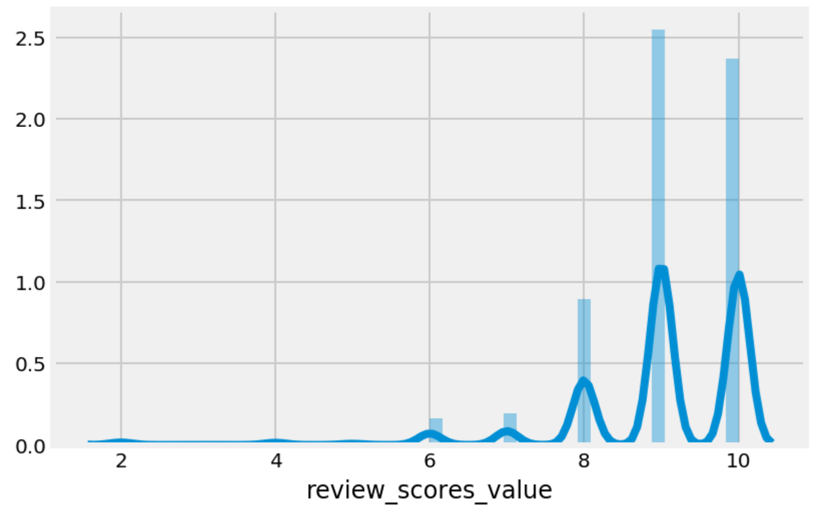
In [85]:
```
# Plot overal reviews scores

sns.distplot(sglisting['review_scores_location'].dropna())
```

Out[85]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a20b86860>
```



In [86]:
```
# Plot overal reviews scores

sns.distplot(sglisting['review_scores_value'].dropna())
```

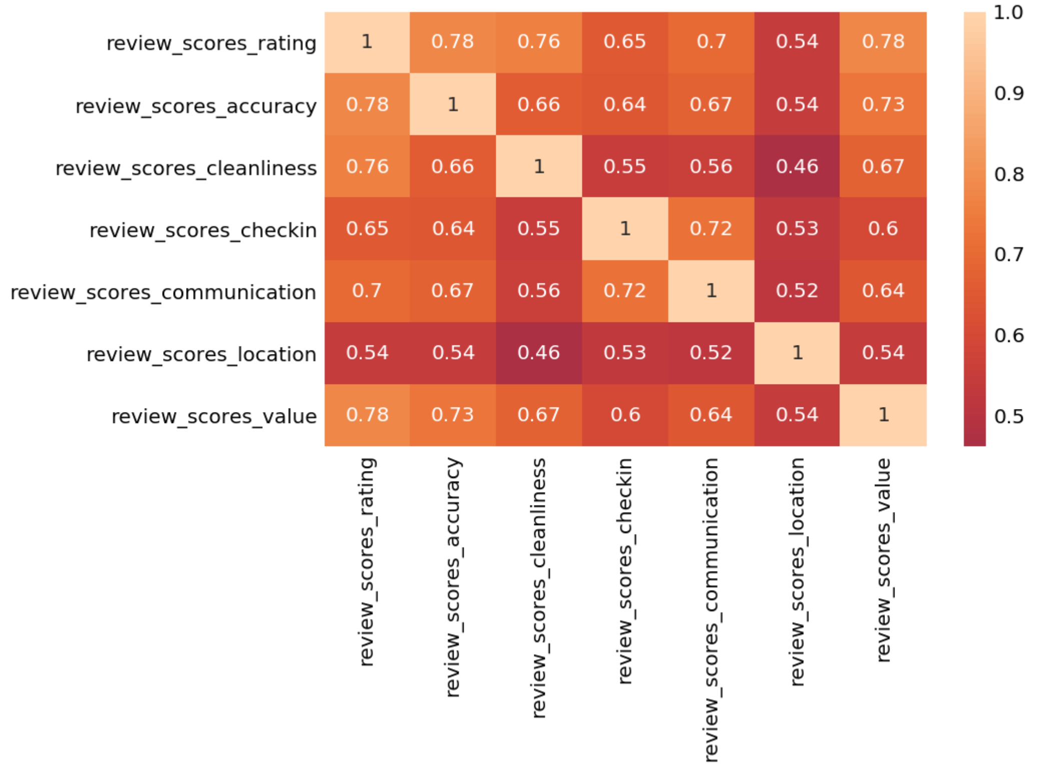
Out[86]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2105abe0>
```



Guests tend to rate value lower than the other portions - customers always demand more value (more benefits for a lower price).

Typesetting math: 0%

In [90]:
```
# Plot correlation between the different scores

sns.heatmap(sglisting[['review_scores_rating','review_scores_accuracy','rev
                      'review_scores_checkin','review_scores_communication'
                      'review_scores_value']].corr(), center=0, annot=True)
```



In [91]:
```
# Find the average score for different features

sglisting[['review_scores_rating','review_scores_accuracy','review_scores_c
                      'review_scores_checkin','review_scores_communication'
                      'review_scores_value']].mean()
```

Out[91]:
```
review_scores_rating           90.700226
review_scores_accuracy          9.252925
review_scores_cleanliness       9.033846
review_scores_checkin           9.523017
review_scores_communication     9.496202
review_scores_location          9.324152
review_scores_value             9.021167
dtype: float64
```

Data Cleaning - Reviews dataset

Drop null reviews

We are unable to obtain any information from the null reviews in this case

Typesetting math: 0%

In [3]:
```
# Drop all null values in sgreviews

sgreviews.dropna(inplace = True)
```

Drop all automated reviews

In [4]:
```
# Find common text present for automated reviews due to cancellation
# Filter them out to remove them in the next step

txt = 'This is an automated posting'

# Get list of actual reviews vs automated reviews
cleaned_review = []
automated_review = []

for value in sgreviews['comments'].values:
    if txt in value:
        automated_review.append(value)
    else:
        cleaned_review.append(value)

# Get list of all reviews, automated reviews as null values
all_reviews_cleaned = []

for value in sgreviews['comments'].values:
    if txt not in value:
        all_reviews_cleaned.append(value)
    else:
        all_reviews_cleaned.append(np.nan)
```

In [5]:
```
 Count number of automated reviews vs all reviews

rint('{}{} {}{}'.format('Number of automated reviews: ',len(automated_revie
rint('{}{} {}{}'.format('Number of actual reviews: ',len(cleaned_review), l
rint('{}{}'.format('Number of total reviews: ',len(automated_review)+len(cl
```

```
Number of automated reviews: 1530 1.5159471697365423%
Number of actual reviews: 99397 98.48405283026345%
Number of total reviews: 100927
```

In [6]:
```
# Create new column for cleaned reviews

sgreviews['all_cleaned'] = all_reviews_cleaned
```

In [7]:
```
# Drop automated reviews

sgreviews.dropna(inplace=True)
```

Check for reviews in foreign languages & Extract only English reviews

Typesetting math: 0%

In [8]:
```
# Import the libary for detecting languages

import langdetect
```

In [9]:
```
# List of languages under langdetect

lang = ['af', 'ar', 'bg', 'bn', 'ca', 'cs', 'cy', 'da', 'de', 'el', 'en', '
        'gu', 'he', 'hi', 'hr', 'hu', 'id', 'it', 'ja', 'kn', 'ko', 'lt', '
        'nl', 'no', 'pa', 'pl', 'pt', 'ro', 'ru', 'sk', 'sl', 'so', 'sq', '
        'tl', 'tr', 'uk', 'ur', 'vi', 'zh']
```

In [10]:
```
cleaned_list = []

for review in sgreviews['all_cleaned'].values:
    cleaned_list.append(review.strip())
```

In [11]:
```
import re

cleaner_list = []

for review in cleaned_list:
    cleaner_list.append(re.sub("[ !@#$%^&*()_+-=.,<>?:;|}{?~`'/1234567890]"

sgreviews['cleaner_list'] = cleaner_list

drop_list = sgreviews[sgreviews['cleaner_list'] == ''].index
sgreviews = sgreviews.drop(drop_list, axis=0)
```

In [12]:
```
# Retreive languages for the reviews using langdetect

lang_list = []

for i,value in enumerate(sgreviews['all_cleaned'].values):
    try:
        lang_list.append(langdetect.detect_langs(value))
    except:
        lang_list.append(langdetect.detect_langs('Undetectable'))
```

In [13]:
```
# Extract the most probable language

cleaned_lang_list = []

for i, value in enumerate(lang_list):
    if value == 'Undetectable':
        cleaned_lang_list.append(np.nan)
    else:
        cleaned_lang_list.append(str(lang_list[i][0])[:2])
```

In [14]:
```
# Add list of languages back into sgreviews dataframe

sgreviews['review_lang'] = cleaned_lang_list
```

Typesetting math: 0%

In [15]:
```
# Assign new variable en_reviews for reviews that are in english

en_reviews = sgreviews[sgreviews['review_lang'] == 'en']
```

Sentiment Analysis on reviews

In [16]:
```
# Create function for sentiment analysis using TextBlob

from textblob import TextBlob, Word

def sentiment_analysis_blob(feature):

    sentiment_blob = []

    # Score text using TextBlob sentiment polarity attribute
    for listing in en_reviews[feature].values:
        try:
            sentiment_blob.append(TextBlob(listing.lower()).sentiment.polar
        except AttributeError:
            sentiment_blob.append(0)

    return sentiment_blob
```

In [17]:
```
# Create function for sentiment analysis using Vader

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

def sentiment_analysis_vader(feature):

    # Score text using Vader SentimentIntensityAnalyzer
    analyzer = SentimentIntensityAnalyzer()
    sentiment_vader = []

    for listing in en_reviews[feature].values:
        try:
            sentiment_vader.append(analyzer.polarity_scores(listing.lower())
        except AttributeError:
            sentiment_vader.append({'compound':0.0})

    sentiment_vader_compound = []

    for listing in sentiment_vader:
        sentiment_vader_compound.append(listing['compound'])

    return sentiment_vader_compound
```

Typesetting math: 0%

In [18]:
```
# Use the function on features

en_reviews['comments_blob'] = sentiment_analysis_blob('all_cleaned')
en_reviews['comments_vader'] = sentiment_analysis_vader('all_cleaned')
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  This is separate from the ipykernel package so we can avoid doing impor
ts until
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:4: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  after removing the cwd from sys.path.
```

In [19]:
```
# Average score from blob and vader

en_reviews['comments_ave'] = (en_reviews['comments_blob'] + en_reviews['com
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  This is separate from the ipykernel package so we can avoid doing impor
ts until
```

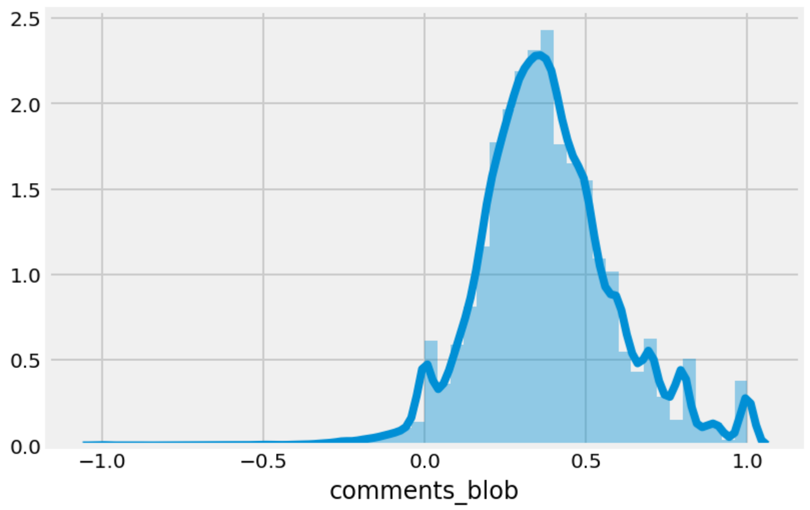
Plot distribution of polarity among ENGLISH reviews

Typesetting math: 0%

In [183]:
```
# Plot distribution of sentiment polarity

sns.distplot(en_reviews['comments_blob'])
```

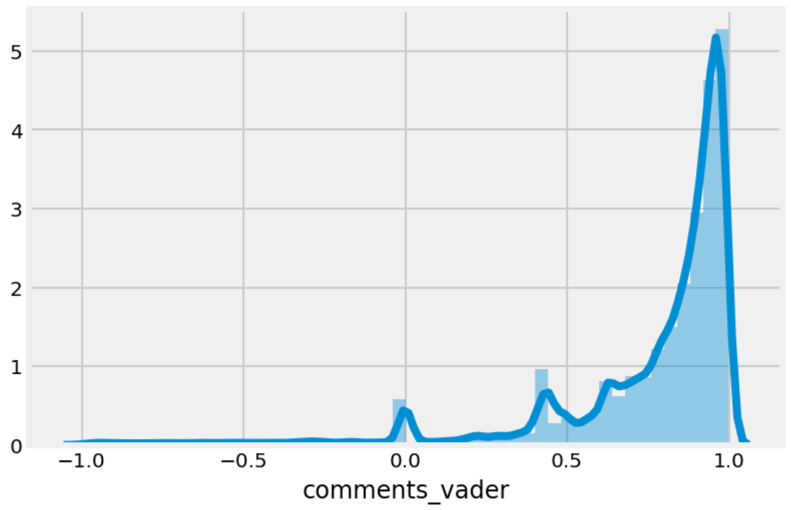
```
/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureW
arning: Using a non-tuple sequence for multidimensional indexing is depre
cated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this wi
ll be interpreted as an array index, `arr[np.array(seq)]`, which will res
ult either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[183]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1be084e0>
```



In [185]:
```
# Plot distribution of sentiment polarity

sns.distplot(en_reviews['comments_vader'])
```

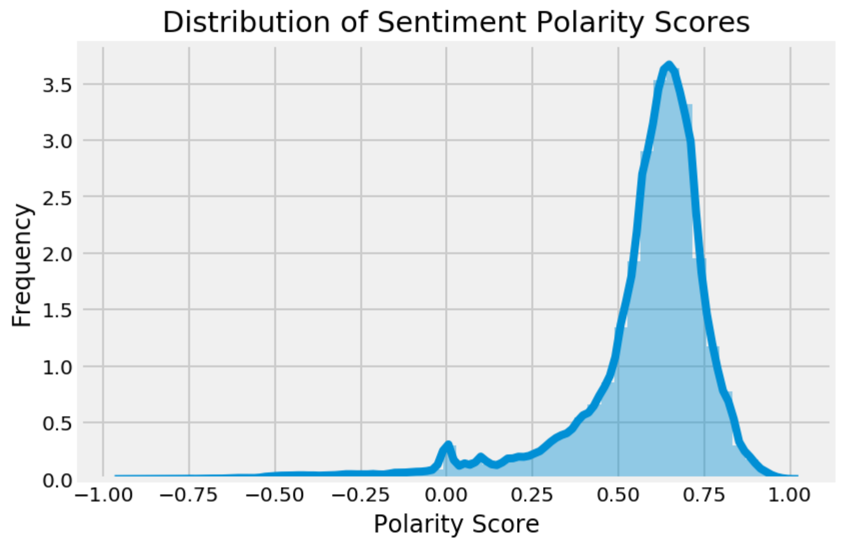
Out[185]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2c486cc0>
```



Typesetting math: 0%

In [65]:
```
# Plot distribution of sentiment polarity

sns.distplot(en_reviews['comments_ave'])
plt.title('Distribution of Sentiment Polarity Scores')
plt.ylabel('Frequency')
plt.xlabel('Polarity Score')
```

Out[65]:
```
Text(0.5, 0, 'Polarity Score')
```



In [299]:
```
# Check the reviews text for the most negative reviews

en_reviews[['all_cleaned','comments_ave']].sort_values(by='comments_ave', a
```

Out[299]:

	all_cleaned	comments_ave
81355	찝쥬이니 져햔테 실슈를 해노코 요히료 염먀를 챠쟈가게때 공찰을 뷰루겠다 흡빡했여오....	-0.926850
91102	It was the worst airbnb experience that I've e...	-0.812450
77437	Worst stay ever	-0.812450
57902	worst experience ever	-0.812450
71357	The worst ever people trying to con people's m...	-0.812450
37694	Very bad! Whill I asking about canceling my bo...	-0.811500
78813	Location is awful, takes quite a while to get ...	-0.775300
19617	Horrible. Will not going there again	-0.771150
74295	Such a bad room. The room was so dirty and inc...	-0.762283
99211	The place is not as advertised. I had lots of ...	-0.754783
30102	Worst and Terrible Host. Not for recommendatio...	-0.751333
15050	poor communication and worst reception.	-0.751000
54149	Wifi very bad	-0.747450
86654	This guy canceled on us WHILE we were looking ...	-0.747450
45086	it is a horrible horrible experience that the ...	-0.744100

Typesetting math: 0%

In [46]:
```
# Average polarity score for the English reviews

en_reviews['comments_ave'].mean()
```

Out[46]:
```
0.5743625682821794
```

Bag of Words on all reviews

In [23]:
```
# Load NLP libraries

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction import stop_words
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from textblob import TextBlob, Word
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
```

In [24]:
```
# Create function to get frequent words for CountVectorizer / TF-IDF Vector

def get_freq_words(sparse_counts, columns):
    # X_all is a sparse matrix, so sum() returns a 'matrix' datatype ...
    #   which we then convert into a 1-D ndarray for sorting
    word_counts = np.asarray(sparse_counts.sum(axis=0)).reshape(-1)

    # argsort() returns smallest first, so we reverse the result
    largest_count_indices = word_counts.argsort()[::-1]

    # pretty-print the results! Remember to always ask whether they make se
    freq_words = pd.Series(word_counts[largest_count_indices],
                           index=columns[largest_count_indices])

    return freq_words
```

In [27]:
```
Retreive default stopwords in the NLTK library

op = stopwords.words('english')

Add more stopwords that are applicable in this case
op.extend(['stay','recommend','recommended','nice','definitely','great','pla
```

In [31]:
```
cvec = CountVectorizer(stop_words=stop, ngram_range=(2,4), min_df=80, max_d
cvec.fit(en_reviews['all_cleaned'])
df_train  = pd.DataFrame(cvec.transform(en_reviews['all_cleaned']).todense(
              columns=cvec.get_feature_names())
df_test = pd.DataFrame(cvec.transform(en_reviews['all_cleaned']).todense(),
              columns=cvec.get_feature_names())
columns = np.array(cvec.get_feature_names())
```

Typesetting math: 0%

In [32]:
```
get_freq_words(cvec.transform(en_reviews['all_cleaned']), columns)
```

Out[32]:
```
mrt station                     5571
walking distance                3528
bus stop                        2731
room clean                      2515
close mrt                       2103
value money                     1936
clean comfortable               1798
public transport                1768
near mrt                        1760
minutes walk                    1737
apartment clean                 1724
swimming pool                   1666
location close                  1524
everything need                 1480
little india                    1477
easy access                     1394
minute walk                     1372
walk mrt                        1366
friendly helpful                1306
easy get                        1277
orchard road                    1244
visit singapore                 1190
convenient location             1190
mins walk                       1173
washing machine                 1160
location near                   1139
gave us                         1134
min walk                        1126
bus stops                       1080
clean well                      1070
                                ...
singapore near                    81
see pictures                      81
friendly always                   81
strategically located             81
like local                        81
location easy find                81
thank hosting                     81
working well                      81
location wonderful                81
places eat nearby                 80
helpful information               80
singapore family                  80
solo traveller                    80
whole apartment                   80
house spacious                    80
pool roof                         80
convenient transportation         80
stop right outside                80
little noisy                      80
clean check                       80
clean facilities                  80
mrt restaurants                   80
room basic                        80
fast respond                      80
```

Typesetting math: 0%

```
room clean well              80
clean lovely                 80
room convenient              80
make sure everything         80
heavy luggage                80
get see                      80
Length: 1866, dtype: int64
```

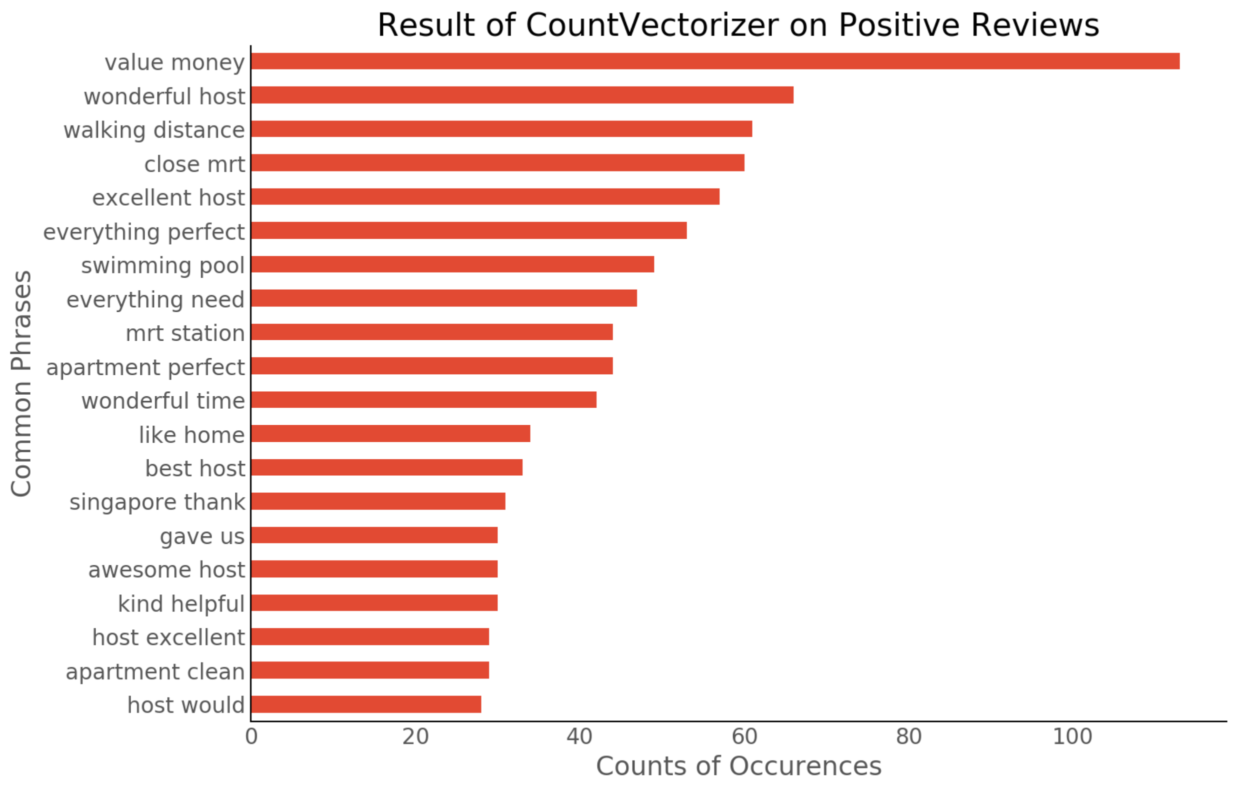
Bag of Words on Positive Reviews

In [134]:
```
Retreive default stopwords in the NLTK library

op = stopwords.words('english')

Add more stopwords that are applicable in this case
op.extend(['stay','recommend','recommended','nice','definitely','great','pla
```

In [135]:
```
cvec = CountVectorizer(stop_words=stop, ngram_range=(2,4), min_df=10, max_d
cvec.fit(en_reviews[en_reviews['comments_ave'] > 0.8]['all_cleaned'])
df_train  = pd.DataFrame(cvec.transform(en_reviews[en_reviews['comments_ave
              columns=cvec.get_feature_names())
df_test = pd.DataFrame(cvec.transform(en_reviews[en_reviews['comments_ave']
              columns=cvec.get_feature_names())
columns = np.array(cvec.get_feature_names())
```

Typesetting math: 0%

In [136]:
```
plt.rcParams.update({'font.size': 14})
fig, ax = plt.subplots(1,1)
get_freq_words(cvec.transform(en_reviews[en_reviews['comments_ave'] > 0.8][
plt.xlabel('Counts of Occurences')
plt.ylabel('Common Phrases')
plt.title('Result of CountVectorizer on Positive Reviews')
ax.spines['bottom'].set_color('black')
ax.spines['left'].set_color('black')
fig.set_facecolor('white')
ax.set_facecolor('white');
```



Typesetting math: 0%

Bag of Words on Negative Reviews

In [104]:
```
# Retreive default stopwords in the NLTK library

stop = stopwords.words('english')

# Add more stopwords that are applicable in this case
stop.extend(['stay','recommend','recommended','nice','definitely','great','
             'website','hidden','airbnb','mrt','bus','mins','bnb','location
            'experience','first'])
```

In [105]:
```
cvec = CountVectorizer(stop_words=stop, ngram_range=(2,4), min_df=10, max_d
cvec.fit(en_reviews[en_reviews['comments_ave'] < 0]['all_cleaned'])
df_train  = pd.DataFrame(cvec.transform(en_reviews[en_reviews['comments_ave
             columns=cvec.get_feature_names())
df_test = pd.DataFrame(cvec.transform(en_reviews[en_reviews['comments_ave']
             columns=cvec.get_feature_names())
columns = np.array(cvec.get_feature_names())
```

Typesetting math: 0%

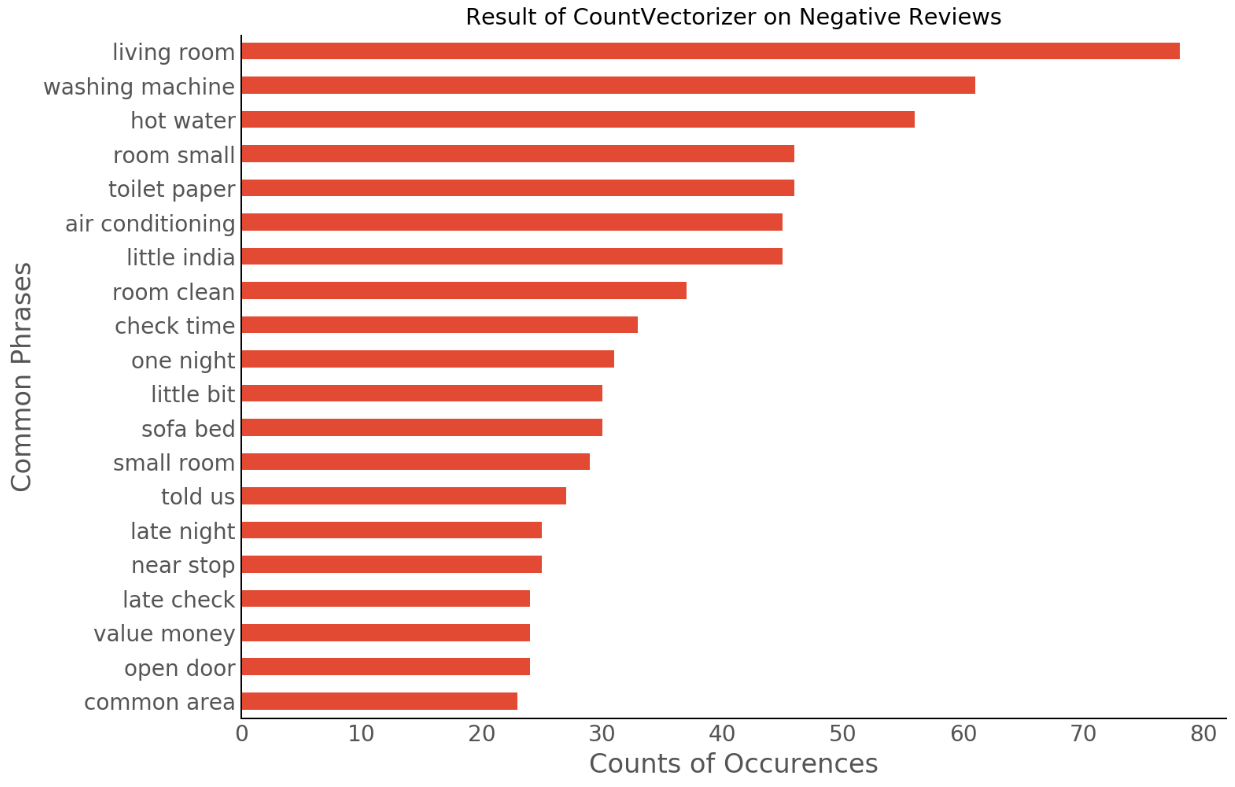
In [106]:
```
get_freq_words(cvec.transform(en_reviews[en_reviews['comments_ave'] < 0]['a
```

Out[106]:
```
living room          78
washing machine      61
hot water            56
room small           46
toilet paper         46
air conditioning     45
little india         45
room clean           37
check time           33
one night            31
little bit           30
sofa bed             30
small room           29
told us              27
late night           25
near stop            25
value money          24
open door            24
late check           24
common area          23
air conditioner      23
clean room           22
one room             22
last minute          21
really bad           20
room bathroom        20
gave us              20
last day             20
one bathroom         19
every time           19
                     ..
quite noisy          11
overall bad          11
one thing            11
one bed              11
never met            11
nearest station      11
one person           11
find another         11
however room         11
farrer park          11
check easy           10
air condition        10
working properly     10
day check            10
washer dryer         10
extremely small      10
arrived late         10
feel like            10
basic amenities      10
bathroom toilet      10
look like            10
size bed             10
room kitchen         10
get refund           10
```

Typesetting math: 0%

```
host said           10
close stop          10
different room      10
living area         10
long time           10
worth money         10
Length: 157, dtype: int64
```

In [128]:
```
plt.rcParams.update({'font.size': 14})
fig, ax = plt.subplots(1,1)
plt.style.use('ggplot')
get_freq_words(cvec.transform(en_reviews[en_reviews['comments_ave'] < 0]['a
plt.xlabel('Counts of Occurences')
plt.ylabel('Common Phrases')
plt.title('Result of CountVectorizer on Negative Reviews')
ax.spines['bottom'].set_color('black')
ax.spines['left'].set_color('black')
fig.set_facecolor('white')
ax.set_facecolor('white');
```



Topic Modelling on Reviews

In [20]:
```
# Import gensim package

import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel
```

Typesetting math: 0%

In [21]:
```
# spacy for lemmatization
import spacy
```

In [33]:
```
# Convert to list
data = en_reviews['all_cleaned'].values.tolist()
```

In [34]:
```
# Tokenize all the reviews (remove punctuations, make lowercase)

def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))  #

data_words = list(sent_to_words(data))

    # Check tokenized reviews
print(data_words[:1])
```

```
[['fran', 'was', 'absolutely', 'gracious', 'and', 'welcoming', 'made', 'm
y', 'stay', 'great', 'experience', 'would', 'definitely', 'recommend', 't
his', 'cozy', 'and', 'peaceful', 'place', 'to', 'anyone']]
```

In [35]:
```
# Build the bigram and trigram models
bigram = gensim.models.Phrases(data_words, min_count=5, threshold=50) # hig
trigram = gensim.models.Phrases(bigram[data_words], threshold=50)

# Faster way to get a sentence clubbed as a trigram/bigram
bigram_mod = gensim.models.phrases.Phraser(bigram)
trigram_mod = gensim.models.phrases.Phraser(trigram)

# See trigram example
print(trigram_mod[bigram_mod[data_words[0]]])
```

```
/anaconda3/lib/python3.6/site-packages/gensim/models/phrases.py:494: User
Warning: For a faster implementation, use the gensim.models.phrases.Phras
er class
  warnings.warn("For a faster implementation, use the gensim.models.phras
es.Phraser class")
/anaconda3/lib/python3.6/site-packages/gensim/models/phrases.py:494: User
Warning: For a faster implementation, use the gensim.models.phrases.Phras
er class
  warnings.warn("For a faster implementation, use the gensim.models.phras
es.Phraser class")
/anaconda3/lib/python3.6/site-packages/gensim/models/phrases.py:494: User
Warning: For a faster implementation, use the gensim.models.phrases.Phras
er class
  warnings.warn("For a faster implementation, use the gensim.models.phras
es.Phraser class")
/anaconda3/lib/python3.6/site-packages/gensim/models/phrases.py:494: User
Warning: For a faster implementation, use the gensim.models.phrases.Phras
er class
  warnings.warn("For a faster implementation, use the gensim.models.phras
```

Typesetting math: 0%

In [36]:
```
# Define functions for stopwords, bigrams, trigrams and lemmatization
def remove_stopword(texts):
    return [[word for word in simple_preprocess(str(doc)) if word not in st

def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in al
    return texts_out
```

In [37]:
```
# Remove Stop Words
data_words_nostops = remove_stopword(data_words)

# Form Bigrams
data_words_bigrams = make_bigrams(data_words_nostops)

# Initialize spacy 'en' model, keeping only tagger component (for efficienc
# python3 -m spacy download en
nlp = spacy.load('en', disable=['parser', 'ner'])

# Do lemmatization keeping only noun, adj, vb, adv
data_lemmatized = lemmatization(data_words_bigrams, allowed_postags=['NOUN'

print(data_lemmatized[:1])
```

```
[['fran', 'absolutely', 'gracious', 'welcome', 'make', 'experience', 'wou
ld', 'cozy', 'peaceful', 'anyone']]
```

In [38]:
```
# Create Dictionary
id2word = corpora.Dictionary(data_lemmatized)

# Create Corpus
texts = data_lemmatized

# Term Document Frequency
corpus = [id2word.doc2bow(text) for text in texts]

# View
print(corpus[:1])
```

```
[[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1),
(9, 1)]]
```

Typesetting math: 0%

In [254]:
```
# Human readable format of corpus (term-frequency)
[[(id2word[id], freq) for id, freq in cp] for cp in corpus[:1]]
```

Out[254]:
```
[[('absolutely', 1),
  ('anyone', 1),
  ('cozy', 1),
  ('experience', 1),
  ('fran', 1),
  ('gracious', 1),
  ('make', 1),
  ('peaceful', 1),
  ('welcome', 1),
  ('would', 1)]]
```

In [271]:
```
# Build LDA model - try with 10 topics
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=10,
                                            random_state=100,
                                            update_every=1,
                                            chunksize=100,
                                            passes=10,
                                            alpha='auto',
                                            per_word_topics=True)
```

```
recationWarning: Calling np.sum(generator) is deprecated, and in the futu
re will give a different result. Use np.sum(np.from_iter(generator)) or t
he python sum builtin instead.
  score += np.sum(cnt * logsumexp(Elogthetad + Elogbeta[:, int(id)]) for
id, cnt in doc)
/anaconda3/lib/python3.6/site-packages/gensim/models/ldamodel.py:826: Dep
recationWarning: Calling np.sum(generator) is deprecated, and in the futu
re will give a different result. Use np.sum(np.from_iter(generator)) or t
he python sum builtin instead.
  score += np.sum(cnt * logsumexp(Elogthetad + Elogbeta[:, int(id)]) for
id, cnt in doc)
/anaconda3/lib/python3.6/site-packages/gensim/models/ldamodel.py:826: Dep
recationWarning: Calling np.sum(generator) is deprecated, and in the futu
re will give a different result. Use np.sum(np.from_iter(generator)) or t
he python sum builtin instead.
  score += np.sum(cnt * logsumexp(Elogthetad + Elogbeta[:, int(id)]) for
id, cnt in doc)
/anaconda3/lib/python3.6/site-packages/gensim/models/ldamodel.py:826: Dep
recationWarning: Calling np.sum(generator) is deprecated, and in the futu
```

Typesetting math: 0%

In [272]:
```
# Print the Keyword in the 10 topics

from pprint import pprint

pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(0,
  '0.099*"check" + 0.047*"night" + 0.045*"super" + 0.041*"even" + '
  '0.034*"excellent" + 0.033*"pool" + 0.029*"lovely" + 0.018*"late" + '
  '0.018*"pleasant" + 0.018*"fantastic"'),
 (1,
  '0.048*"bed" + 0.040*"small" + 0.039*"bathroom" + 0.033*"value" + '
  '0.025*"kitchen" + 0.025*"money" + 0.023*"bedroom" + 0.022*"guest" + '
  '0.022*"unit" + 0.017*"respond"'),
 (2,
  '0.105*"time" + 0.062*"amazing" + 0.044*"expect" + 0.041*"book" + '
  '0.035*"wonderful" + 0.026*"friend" + 0.025*"stylish" + 0.025*"new" + '
  '0.020*"condo" + 0.017*"better"'),
 (3,
  '0.069*"get" + 0.041*"go" + 0.038*"day" + 0.033*"find" + 0.032*"use" +
'
  '0.032*"airbnb" + 0.029*"good" + 0.022*"want" + 0.022*"could" + '
  '0.021*"thing"'),
 (4,
  '0.195*"apartment" + 0.092*"well" + 0.033*"work" + 0.026*"picture" + '
  '0.024*"building" + 0.022*"awesome" + 0.015*"show" + 0.015*"property" +
'
  '0.014*"kid" + 0.012*"communicate"'),
 (5,
  '0.084*"location" + 0.051*"mrt" + 0.031*"easy" + 0.030*"close" + '
  '0.030*"station" + 0.028*"walk" + 0.025*"convenient" + 0.022*"food" + '
  '0.019*"area" + 0.017*"lot"'),
 (6,
  '0.084*"clean" + 0.076*"host" + 0.067*"room" + 0.035*"really" + '
  '0.035*"would" + 0.029*"thank" + 0.027*"comfortable" + 0.027*"need" + '
  '0.024*"everything" + 0.021*"helpful"'),
 (7,
  '0.070*"responsive" + 0.050*"response" + 0.039*"available" + 0.030*"wor
th" + '
  '0.028*"enjoy" + 0.027*"comfy" + 0.025*"able" + 0.023*"cleanliness" + '
  '0.022*"reply" + 0.021*"free"'),
 (8,
  '0.038*"bit" + 0.037*"little" + 0.036*"price" + 0.027*"door" + '
  '0.026*"however" + 0.023*"may" + 0.022*"hotel" + 0.017*"bad" + 0.017*"t
hink" '
  '+ 0.014*"open"'),
 (9,
  '0.155*"singapore" + 0.060*"make" + 0.050*"home" + 0.042*"feel" + '
  '0.038*"place" + 0.030*"beautiful" + 0.028*"trip" + 0.026*"visit" + '
  '0.020*"hospitality" + 0.020*"welcome"')]
```

Typesetting math: 0%

In [273]:
```
# Compute Perplexity for 10 topics
print('\nPerplexity: ', lda_model.log_perplexity(corpus))
    # a measure of how good the model is. lower the better.

# Compute Coherence Score for 10 topics
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
he python sum builtin instead.
  score += np.sum(cnt * logsumexp(Elogthetad + Elogbeta[:, int(id)]) for
id, cnt in doc)
/anaconda3/lib/python3.6/site-packages/gensim/models/ldamodel.py:826: Dep
recationWarning: Calling np.sum(generator) is deprecated, and in the futu
re will give a different result. Use np.sum(np.from_iter(generator)) or t
he python sum builtin instead.
  score += np.sum(cnt * logsumexp(Elogthetad + Elogbeta[:, int(id)]) for
id, cnt in doc)
/anaconda3/lib/python3.6/site-packages/gensim/models/ldamodel.py:826: Dep
recationWarning: Calling np.sum(generator) is deprecated, and in the futu
re will give a different result. Use np.sum(np.from_iter(generator)) or t
he python sum builtin instead.
  score += np.sum(cnt * logsumexp(Elogthetad + Elogbeta[:, int(id)]) for
id, cnt in doc)


Perplexity:  -7.362204337574667

Coherence Score:  0.5647060745372311
```

Visualizing Topic Modelling

In [44]:
```
# Plotting tools
import pyLDAvis
import pyLDAvis.gensim  # don't skip this
import matplotlib.pyplot as plt
%matplotlib inline
```

In [276]:
```
# Visualize the topics for 10 topics
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
vis
```

```
/anaconda3/lib/python3.6/site-packages/pyLDAvis/_prepare.py:257: FutureWa
rning: Sorting because non-concatenation axis is not aligned. A future ve
rsion
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  return pd.concat([default_term_info] + list(topic_dfs))
```

Out[276]: Typesetting math: 0%

Refine the model by trying different number of topics

Try different number of topics - aim to get the best perplexity and coherence scores, and where the topics mapped out during visualization do not overlap each other and are as far away from each other as possible.

In [39]:
```
# Build LDA model for 5 topics
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=5,
                                            random_state=100,
                                            update_every=1,
                                            chunksize=100,
                                            passes=10,
                                            alpha='auto',
                                            per_word_topics=True)
```

In [40]:
```
# Print the Keyword in the 5 topics

from pprint import pprint

pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(0,
  '0.084*"apartment" + 0.039*"well" + 0.029*"area" + 0.024*"perfect" + '
  '0.022*"amazing" + 0.018*"view" + 0.017*"locate" + 0.017*"excellent" +
'
  '0.017*"pool" + 0.016*"communication"'),
 (1,
  '0.019*"get" + 0.017*"use" + 0.016*"day" + 0.014*"night" + 0.012*"work"
+ '
  '0.012*"could" + 0.011*"thing" + 0.010*"guest" + 0.010*"airbnb" + '
  '0.009*"sleep"'),
 (2,
  '0.111*"room" + 0.035*"bed" + 0.029*"small" + 0.029*"bathroom" + 0.024
*"big" '
  '+ 0.022*"space" + 0.021*"overall" + 0.020*"quite" + 0.018*"expect" + '
  '0.018*"people"'),
 (3,
  '0.098*"location" + 0.059*"mrt" + 0.036*"easy" + 0.035*"close" + '
  '0.035*"station" + 0.033*"walk" + 0.029*"convenient" + 0.026*"food" + '
  '0.020*"lot" + 0.019*"bus"'),
 (4,
  '0.055*"clean" + 0.050*"host" + 0.028*"singapore" + 0.023*"really" + '
  '0.023*"check" + 0.023*"would" + 0.019*"thank" + 0.018*"comfortable" +
'
  '0.018*"need" + 0.017*"time"')]
```

Typesetting math: 0%

In [41]:
```
# Compute Perplexity for 5 topics
print('\nPerplexity: ', lda_model.log_perplexity(corpus))
    # a measure of how good the model is. lower the better.

# Compute Coherence Score for 5 topics
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```

Perplexity:  -7.210659748379875

Coherence Score:  0.6214173456128512
```

In [45]:
```
# Visualize the topics for 5 topics
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
vis
```

```
/anaconda3/lib/python3.6/site-packages/pyLDAvis/_prepare.py:257: FutureWa
rning: Sorting because non-concatenation axis is not aligned. A future ve
rsion
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  return pd.concat([default_term_info] + list(topic_dfs))
```

Out[45]:

Typesetting math: 0%

In [281]:
```
# Build LDA model for 4 topics
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                           id2word=id2word,
                                           num_topics=4,
                                           random_state=100,
                                           update_every=1,
                                           chunksize=100,
                                           passes=10,
                                           alpha='auto',
                                           per_word_topics=True)
```

```
The history saving thread hit an unexpected error (OperationalError('unab
le to open database file',)).History will not be written to the database.
```

In [282]:
```
# Print the Keyword in the 4 topics

from pprint import pprint

pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

```
[(0,
  '0.052*"mrt" + 0.031*"close" + 0.031*"station" + 0.029*"walk" + '
  '0.025*"convenient" + 0.023*"food" + 0.018*"area" + 0.017*"bus" + '
  '0.017*"restaurant" + 0.015*"minute"'),
 (1,
  '0.060*"room" + 0.019*"bed" + 0.016*"small" + 0.015*"bathroom" + 0.013
*"use" '
  '+ 0.012*"night" + 0.010*"get" + 0.010*"kitchen" + 0.010*"money" + '
  '0.009*"work"'),
 (2,
  '0.044*"singapore" + 0.035*"really" + 0.029*"thank" + 0.026*"time" + '
  '0.020*"family" + 0.019*"stay" + 0.019*"go" + 0.019*"house" + '
  '0.017*"perfect" + 0.017*"make"'),
 (3,
  '0.061*"location" + 0.058*"clean" + 0.053*"host" + 0.040*"apartment" +
'
  '0.024*"check" + 0.024*"would" + 0.019*"well" + 0.019*"need" + '
  '0.018*"comfortable" + 0.017*"everything"')]
```

Typesetting math: 0%

In [283]:
```
# Compute Perplexity for 4 topics
print('\nPerplexity: ', lda_model.log_perplexity(corpus))
    # a measure of how good the model is. lower the better.

# Compute Coherence Score for 4 topics
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
/anaconda3/lib/python3.6/site-packages/gensim/models/ldamodel.py:826: Dep
recationWarning: Calling np.sum(generator) is deprecated, and in the futu
re will give a different result. Use np.sum(np.from_iter(generator)) or t
he python sum builtin instead.
  score += np.sum(cnt * logsumexp(Elogthetad + Elogbeta[:, int(id)]) for
id, cnt in doc)
/anaconda3/lib/python3.6/site-packages/gensim/models/ldamodel.py:826: Dep
recationWarning: Calling np.sum(generator) is deprecated, and in the futu
re will give a different result. Use np.sum(np.from_iter(generator)) or t
he python sum builtin instead.
  score += np.sum(cnt * logsumexp(Elogthetad + Elogbeta[:, int(id)]) for
id, cnt in doc)
/anaconda3/lib/python3.6/site-packages/gensim/models/ldamodel.py:826: Dep
recationWarning: Calling np.sum(generator) is deprecated, and in the futu
re will give a different result. Use np.sum(np.from_iter(generator)) or t
he python sum builtin instead.
  score += np.sum(cnt * logsumexp(Elogthetad + Elogbeta[:, int(id)]) for
id, cnt in doc)
/anaconda3/lib/python3.6/site-packages/gensim/models/ldamodel.py:826: Dep
```

In [284]:
```
# Visualize the topics for 4 topics
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
vis
```

```
/anaconda3/lib/python3.6/site-packages/pyLDAvis/_prepare.py:257: FutureWa
rning: Sorting because non-concatenation axis is not aligned. A future ve
rsion
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  return pd.concat([default_term_info] + list(topic_dfs))
```

Out[284]:

Typesetting math: 0%