Oops... Nothing Here.. So, you are on your own this time.

Data

In [98]:
```
!pip install wget
!pip install twython
import wget
```

```
Requirement already satisfied: wget in /usr/local/lib/python3.6/dist-pa
ckages (3.2)
Requirement already satisfied: twython in /usr/local/lib/python3.6/dist
-packages (3.7.0)
Requirement already satisfied: requests-oauthlib>=0.4.0 in /usr/local/l
ib/python3.6/dist-packages (from twython) (1.3.0)
Requirement already satisfied: requests>=2.1.0 in /usr/local/lib/python
3.6/dist-packages (from twython) (2.21.0)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python
3.6/dist-packages (from requests-oauthlib>=0.4.0->twython) (3.1.0)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/
python3.6/dist-packages (from requests>=2.1.0->twython) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/pyt
hon3.6/dist-packages (from requests>=2.1.0->twython) (2019.9.11)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/
python3.6/dist-packages (from requests>=2.1.0->twython) (1.24.3)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python
3.6/dist-packages (from requests>=2.1.0->twython) (2.8)
```

In [99]:
```
#Plot
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
#Data Packages
import math
import pandas
import numpy as np

#Progress bar
!pip install tqdm
from tqdm import tqdm

#Counter
from collections import Counter

#Operation
import operator

#Natural Language Processing Packages
import re
import nltk
nltk.download('punkt')
## Download Resources
nltk.download("vader_lexicon")#identify emotion rule-based not learned
nltk.download("stopwords")
nltk.download("averaged_perceptron_tagger")
nltk.download("wordnet")

from nltk.sentiment import SentimentAnalyzer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.tokenize import RegexpTokenizer
from nltk.sentiment.util import *
from nltk import tokenize
from nltk.corpus import stopwords
from nltk.tag import PerceptronTagger
from nltk.data import find
import pandas as pd
## Machine Learning
import sklearn
import sklearn.metrics as metrics
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-pa
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-pa
ckages (4.28.1)
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [0]:
```
reviewDF=pd.read_csv("reviews.csv")
reviewDF.columns=['file','hotel name','review','rating','sentiment']
```

In [101]:
```
reviewDF.head()
```

Out[101]:

	file	hotel name	review	rating	sentiment
0	data\ca\155019\155495\682627838.html	Fairmont Royal York	"I was there to celebrate my daughters 30th Bi...	5	positive
1	data\ca\155019\155495\682656607.html	Fairmont Royal York	"Very central location, very sizeable hotel, b...	3	negative
2	data\ca\155019\155495\682676167.html	Fairmont Royal York	"We got to the hotel pretty late, around 6pm a...	5	positive

	file	hotel name	review	rating	sentiment
3	data\ca\155019\155495\682872848.html	Fairmont Royal York	"My family had the most relaxed stay at Fairmo...	5	positive
4	data\ca\155019\155495\683051089.html	Fairmont Royal York	"We came to the Royal York to attend our son's...	1	negative

In [102]:
```
reviewDF['hotel name'].unique()
```

Out[102]:
```
array(['Fairmont Royal York', 'Hotel Victoria', 'Hyatt Regency Toronto',
       'InterContinental Toronto Centre', 'Cambridge Suites Toronto',
       'Sheraton Centre Toronto Hotel', 'Hilton Toronto',
       'The Strathcona Hotel', 'The Ritz-Carlton, Toronto',
       'The St. Regis Toronto', 'Le Germain Hotel Toronto Mercer',
       'Shangri-La Hotel Toronto', 'The Beverley Hotel',
       'One King West Hotel &amp;amp; Residence',
       'Residence Inn Toronto Downtown/Entertainment District',
       'Delta Hotels by Marriott Toronto',
       'Holiday Inn Express Hotel &amp;amp; Suites',
       'Novotel Toronto Vaughan Centre',
       'Extended Stay Canada - Toronto - Vaughan',
       'Homewood Suites by Hilton Toronto Vaughan',
       'SpringHill Suites Toronto Vaughan', 'Aloft Vaughan Mills',
       'Courtyard Toronto Vaughan', 'Element Vaughan Southwest',
       'Monte Carlo Inn Vaughan Suites',
       'Hilton Garden Inn Toronto/Vaughan',
       'Residence Inn Toronto Vaughan', 'Liberty Suites Hotel',
       'Super 8 by Wyndham Toronto North', 'Vaughan Inn'], dtype=object)
```

Q1. Sentiment Analysis and Aggregation

(a)

In [0]:
```
def evalText(text, to_df=False, columns=[]):
  sid = SentimentIntensityAnalyzer()
  pdlist = []
  if to_df:
      for sentence in tqdm(text):
          ss = sid.polarity_scores(sentence)
          pdlist.append([sentence]+[ss['compound']])
      reviewDf = pandas.DataFrame(pdlist)
      reviewDf.columns = columns
      return reviewDf

  else:
      for sentence in tqdm(text):
          print(sentence)
          ss = sid.polarity_scores(sentence)
          for k in sorted(ss):
              print('{0}: {1}, '.format(k, ss[k]), end='')
          print()
```

In [104]:
```
reviewText=reviewDF['review'].values
vaderDF=evalText(reviewText, to_df=True, columns=['reviewCol','vader'])
```

```
100%|██████████| 5157/5157 [00:06<00:00, 788.45it/s]
```

In [105]:
```
vaderDF.head()
```

Out[105]:

	reviewCol	vader
0	"I was there to celebrate my daughters 30th Bi...	0.9818
1	"Very central location, very sizeable hotel, b...	0.8370
2	"We got to the hotel pretty late, around 6pm a...	0.8500
3	"My family had the most relaxed stay at Fairmo...	0.9785

	reviewCol	vader
4	"We came to the Royal York to attend our son's...	-0.3493

In [0]:
```
reviewDF['vader']=vaderDF['vader']
```

In [107]:
```
reviewDF.head()
```

Out[107]:

	file	hotel name	review	rating	sentiment	vader
0	data\ca\155019\155495\682627838.html	Fairmont Royal York	"I was there to celebrate my daughters 30th Bi...	5	positive	0.9818
1	data\ca\155019\155495\682656607.html	Fairmont Royal York	"Very central location, very sizeable hotel, b...	3	negative	0.8370
2	data\ca\155019\155495\682676167.html	Fairmont Royal York	"We got to the hotel pretty late, around 6pm a...	5	positive	0.8500

	file	hotel name	review	rating	sentiment	vader
3	data\ca\155019\155495\682872848.html	Fairmont Royal York	"My family had the most relaxed stay at Fairmo...	5	positive	0.9785
4	data\ca\155019\155495\683051089.html	Fairmont Royal York	"We came to the Royal York to attend our son's...	1	negative	-0.3493

In [108]:
```
hotelList=reviewDF['hotel name'].unique()
print(type(hotelList))
```

```
<class 'numpy.ndarray'>
```

In [0]:
```
def averageVader(reviewDF,hotelList):
  results={}
  for item in hotelList:
    itemMean=reviewDF.loc[reviewDF['hotel name']==item,'vader'].mean()
    results[item]=itemMean
  return results

def averageGroundTruth(reviewDF,hotelList):
  results={}
  for item in hotelList:
    itemMean=reviewDF.loc[reviewDF['hotel name']==item,'rating'].mean()
    results[item]=itemMean
  return results
```

In [110]:
```
meanVader=averageVader(reviewDF,hotelList)
sorted_meanVader = sorted(meanVader.items(), key=operator.itemgetter(1
))
print(sorted_meanVader)
```

```
[('Super 8 by Wyndham Toronto North', 0.4414619999999999), ('Monte Carl
o Inn Vaughan Suites', 0.633039), ('Hilton Garden Inn Toronto/Vaughan',
0.6620854999999995), ('Hyatt Regency Toronto', 0.6840360000000003), ('L
iberty Suites Hotel', 0.6930644628099172), ('Courtyard Toronto Vaugha
n', 0.7025479999999998), ('Vaughan Inn', 0.7168125), ('Extended Stay Ca
nada - Toronto - Vaughan', 0.7299289999999999), ('The Beverley Hotel',
0.7426484210526315), ('SpringHill Suites Toronto Vaughan', 0.7446214999
999999), ('Novotel Toronto Vaughan Centre', 0.7572675), ('Sheraton Cent
re Toronto Hotel', 0.7688285), ('Hilton Toronto', 0.7773699999999997),
('Residence Inn Toronto Downtown/Entertainment District', 0.78191400000
00003), ('Cambridge Suites Toronto', 0.7832079999999999), ('Residence I
nn Toronto Vaughan', 0.7840546762589926), ('InterContinental Toronto Ce
ntre', 0.7937050000000005), ('Homewood Suites by Hilton Toronto Vaugha
n', 0.8027194999999999), ('Fairmont Royal York', 0.8068414201183436),
('Hotel Victoria', 0.8172959999999999), ('Holiday Inn Express Hotel &am
p; Suites', 0.821143975903614), ('Shangri-La Hotel Toronto', 0.82917049
99999989), ('Element Vaughan Southwest', 0.8418779569892472), ('The Str
athcona Hotel', 0.8499257142857143), ('Le Germain Hotel Toronto Merce
r', 0.8510055000000001), ('One King West Hotel &amp;amp; Residence', 0.8595
835294117646), ('Aloft Vaughan Mills', 0.8602324999999996), ('The Ritz-
Carlton, Toronto', 0.8614105000000002), ('Delta Hotels by Marriott Toro
nto', 0.8679365000000006), ('The St. Regis Toronto', 0.871491499999999
9)]
```

In [111]:
```
meanTruth=averageGroundTruth(reviewDF,hotelList)
sorted_meanTruth = sorted(meanTruth.items(), key=operator.itemgetter(1
))
print(sorted_meanTruth)
```

```
[('Super 8 by Wyndham Toronto North', 3.335), ('The Beverley Hotel', 3.
610526315789474), ('Courtyard Toronto Vaughan', 3.72), ('Monte Carlo In
n Vaughan Suites', 3.79), ('Vaughan Inn', 3.8125), ('Hilton Garden Inn
Toronto/Vaughan', 3.865), ('Liberty Suites Hotel', 3.9338842975206614),
('Hilton Toronto', 4.045), ('Extended Stay Canada - Toronto - Vaughan',
4.065), ('Cambridge Suites Toronto', 4.1), ('Hyatt Regency Toronto', 4.
```

```
155), ('Residence Inn Toronto Downtown/Entertainment District', 4.16),
('Sheraton Centre Toronto Hotel', 4.17), ('Residence Inn Toronto Vaugha
n', 4.18705035971223), ('Novotel Toronto Vaughan Centre', 4.21), ('Spri
ngHill Suites Toronto Vaughan', 4.21), ('Hotel Victoria', 4.26666666666
6667), ('Holiday Inn Express Hotel &amp;amp; Suites', 4.307228915662651),
('InterContinental Toronto Centre', 4.385), ('Fairmont Royal York', 4.4
14201183431953), ('Homewood Suites by Hilton Toronto Vaughan', 4.42),
('Element Vaughan Southwest', 4.456989247311828), ('Aloft Vaughan Mill
s', 4.475), ('The Strathcona Hotel', 4.514285714285714), ('Delta Hotels
by Marriott Toronto', 4.585), ('The St. Regis Toronto', 4.64), ('Le Ger
main Hotel Toronto Mercer', 4.64), ('The Ritz-Carlton, Toronto', 4.69),
('One King West Hotel &amp;amp; Residence', 4.694117647058824), ('Shangri-L
a Hotel Toronto', 4.745)]
```

(b)

In [112]:
```
print(sorted_meanVader[-5:])
print(sorted_meanTruth[-5:])
print(sorted_meanVader[:5])
print(sorted_meanTruth[:5])
```

```
[('One King West Hotel &amp;amp; Residence', 0.8595835294117646), ('Aloft V
aughan Mills', 0.8602324999999996), ('The Ritz-Carlton, Toronto', 0.861
4105000000002), ('Delta Hotels by Marriott Toronto', 0.867936500000000
6), ('The St. Regis Toronto', 0.8714914999999999)]
[('The St. Regis Toronto', 4.64), ('Le Germain Hotel Toronto Mercer',
4.64), ('The Ritz-Carlton, Toronto', 4.69), ('One King West Hotel &amp;amp;
Residence', 4.694117647058824), ('Shangri-La Hotel Toronto', 4.745)]
[('Super 8 by Wyndham Toronto North', 0.4414619999999999), ('Monte Carl
o Inn Vaughan Suites', 0.633039), ('Hilton Garden Inn Toronto/Vaughan',
0.6620854999999995), ('Hyatt Regency Toronto', 0.6840360000000003), ('L
iberty Suites Hotel', 0.6930644628099172)]
[('Super 8 by Wyndham Toronto North', 3.335), ('The Beverley Hotel', 3.
610526315789474), ('Courtyard Toronto Vaughan', 3.72), ('Monte Carlo In
n Vaughan Suites', 3.79), ('Vaughan Inn', 3.8125)]
```

Some of the rankings aggree with each other, but not all of them.

Q2

(a)

In [0]:
```
reviewSeries=reviewDF['review']
stop_words = set(stopwords.words('english'))
reviewSeries=reviewSeries.apply(lambda data: nltk.word_tokenize(data))
reviewSeries=reviewSeries.apply(lambda x: [item for item in x if item n
ot in stop_words])
reviewSeries=reviewSeries.apply(lambda x: [item for item in x if item.i
salpha()])
reviewSeries=reviewSeries.apply(lambda x: ' '.join(x))
reviewSeries=reviewSeries.str.lower()
```

In [0]:
```
reviewDF['review']=reviewSeries
```

In [0]:
```
pos_Reviews=reviewDF.loc[reviewDF['rating']>3,'review']
neg_Reviews=reviewDF.loc[reviewDF['rating']<=3,'review']
```

In [116]:
```
print(pos_Reviews.shape,neg_Reviews.shape,reviewDF.shape)
```

```
(4207,) (950,) (5157, 6)
```

In [117]:
```
reviewDF.head()
```

Out[117]:

	file	hotel name	review	rating	sentiment	vader

	file	hotel name	review	rating	sentiment	vader
0	data\ca\155019\155495\682627838.html	Fairmont Royal York	i celebrate daughters birthday they wonderful ...	5	positive	0.9818
1	data\ca\155019\155495\682656607.html	Fairmont Royal York	very central location sizeable hotel bit old f...	3	negative	0.8370
2	data\ca\155019\155495\682676167.html	Fairmont Royal York	we got hotel pretty late around stoping niagar...	5	positive	0.8500
3	data\ca\155019\155495\682872848.html	Fairmont Royal York	my family relaxed stay fairmont hotel staff en...	5	positive	0.9785
4	data\ca\155019\155495\683051089.html	Fairmont Royal York	we came royal york attend son wedding held sat...	1	negative	-0.3493

In [0]:
```
def getMostKwords(reviewDF,k):
  counter=Counter()
  for item in reviewDF:
    #print(item)
    counter.update([word for word in re.findall(r'\w+', item)])
  topList=[]
  topk=counter.most_common(k)
  #print(topk)
  for item in topk:
    topList.append(item[0])

  return topList
```

In [0]:
```
neg_Reviews=neg_Reviews.values

pos_Reviews=pos_Reviews.values
neg_list=getMostKwords(neg_Reviews,50)
pos_list=getMostKwords(pos_Reviews,50)
```

In [120]:
```
print(neg_list)
print(pos_list)
```

```
['i', 'room', 'hotel', 'the', 'would', 'stay', 'we', 'one', 'staff', 'n
ight', 'breakfast', 'rooms', 'desk', 'front', 'good', 'us', 'get', 'nic
e', 'could', 'like', 'stayed', 'bed', 'time', 'told', 'back', 'servic
e', 'also', 'clean', 'location', 'even', 'it', 'great', 'well', 'smal
l', 'area', 'said', 'check', 'this', 'day', 'floor', 'go', 'two', 'bath
room', 'asked', 'really', 'people', 'booked', 'first', 'toronto', 'nex
t']
['i', 'hotel', 'the', 'room', 'great', 'staff', 'stay', 'we', 'clean',
'breakfast', 'good', 'rooms', 'would', 'toronto', 'nice', 'location',
'service', 'stayed', 'friendly', 'well', 'us', 'comfortable', 'one', 'a
rea', 'also', 'it', 'time', 'desk', 'pool', 'front', 'helpful', 'nigh
t', 'close', 'this', 'really', 'excellent', 'bed', 'could', 'get', 'def
initely', 'everything', 'like', 'recommend', 'food', 'back', 'place',
'restaurant', 'check', 'walk', 'parking']
```

Some words appeared both in positive and negative list, I suppose that may be caused by sometimes customers judge the service of the hotel base on these common terms, like if they like the room the hotel they probably mention that in their positive review, but if they dislike the room they are likely to mention that in reviews as well, so the two list shared some common words for ranking a hotel.

(b)

In [0]:
```
def topkNP_Extraction(review,k):
  grammar = "NP: {<DT>?<JJ>*<NN>}"
  parser = nltk.RegexpParser(grammar)
  tokenizer=RegexpTokenizer(r'\w+')
  noun_phrases_list =[]
  for item in review:
    words = tokenizer.tokenize(item)
    pos_words=nltk.pos_tag(words)
    result = parser.parse(pos_words)
    for subtree in result.subtrees(filter = lambda t: t.label()=='NP'):
      npString=[]
      if len(subtree.leaves())==1:
        continue
      for leaf in subtree.leaves():
        npString.append(leaf[0])
      npWord=' '.join(npString)
      noun_phrases_list.append(npWord)
  counter=Counter()
  counter.update(noun_phrases_list)
  topk=counter.most_common(k)
  return topk
```

In [0]:
```
posNP_List=topkNP_Extraction(pos_Reviews,50)
```

In [123]:
```
posNP_List
```

Out[123]:
```
[('the hotel', 533),
 ('the room', 455),
```

```
('the staff', 403),
('this hotel', 235),
('great location', 227),
('the location', 181),
('the breakfast', 139),
('front desk', 125),
('friendly staff', 119),
('the pool', 106),
('great hotel', 105),
('the bed', 98),
('the service', 96),
('first time', 86),
('the bathroom', 83),
('great service', 82),
('hot tub', 81),
('the front', 80),
('the lobby', 78),
('free breakfast', 72),
('good location', 68),
('nice hotel', 64),
('top notch', 64),
('next time', 62),
('great place', 60),
('great staff', 59),
('the food', 57),
('clean staff', 57),
('canada wonderland', 57),
('easy access', 56),
('excellent service', 50),
('every time', 50),
('short walk', 49),
('full kitchen', 49),
('all staff', 46),
('great room', 45),
('great breakfast', 44),
('excellent location', 42),
('good size', 42),
('perfect location', 40),
('great view', 40),
```

```
 ('great stay', 39),
 ('the restaurant', 38),
 ('recommend hotel', 37),
 ('good breakfast', 36),
 ('stay hotel', 35),
 ('good hotel', 34),
 ('comfortable bed', 34),
 ('good value', 34),
 ('free wifi', 34)]
```

In [0]:
```
negNP_List=topkNP_Extraction(neg_Reviews,50)
```

In [125]:
```
negNP_List
```

Out[125]:
```
[('front desk', 101),
 ('the hotel', 97),
 ('the room', 92),
 ('the staff', 66),
 ('this hotel', 44),
 ('great location', 36),
 ('the location', 29),
 ('another room', 27),
 ('the front', 27),
 ('the bathroom', 25),
 ('next time', 23),
 ('the pool', 23),
 ('another hotel', 22),
 ('hot water', 22),
 ('the bed', 21),
 ('the breakfast', 21),
 ('hot tub', 21),
 ('next day', 20),
 ('nice hotel', 19),
 ('good location', 19),
 ('free breakfast', 19),
 ('speak manager', 18),
 ('next door', 17),
 ('first time', 16),
 ('every time', 16),
```

```
 ('the service', 14),
 ('general manager', 14),
 ('stay hotel', 13),
 ('the lobby', 12),
 ('big deal', 12),
 ('the restaurant', 11),
 ('clean room', 11),
 ('friendly staff', 11),
 ('no one', 11),
 ('hard time', 11),
 ('new room', 11),
 ('late night', 11),
 ('long time', 10),
 ('good thing', 10),
 ('last night', 10),
 ('the next day', 9),
 ('the price', 9),
 ('high tea', 9),
 ('next morning', 8),
 ('clean staff', 8),
 ('second night', 8),
 ('the toilet', 8),
 ('second time', 8),
 ('pool area', 8),
 ('the check', 8)]
```

From the noun phrases we can more clearly see the trend in the reviews, but interesting some 'positive phrases' tends to appear in negative review, for example we can see 'great location' and 'nice hotel' in the negative phrase and by looking into the reviews I found that sometimes people comment things like 'It is a nice hotel but....' or 'Although it has a great location, however...'. Thus, I understand that we cannot simply judge the sentiment from some simply words or phrases but also need to consider the words or phrases around them.

In [0]:

Q3

(a)

In [0]:
```
reviewDF['sentiment']=reviewDF['sentiment'].replace('positive',1)
reviewDF['sentiment']=reviewDF['sentiment'].replace('negative',0)
```

In [127]:
```
reviewDF.head()
```

Out[127]:

	file	hotel name	review	rating	sentiment	vader
0	data\ca\155019\155495\682627838.html	Fairmont Royal York	i celebrate daughters birthday they wonderful ...	5	1	0.9818
1	data\ca\155019\155495\682656607.html	Fairmont Royal York	very central location sizeable hotel bit old f...	3	0	0.8370
2	data\ca\155019\155495\682676167.html	Fairmont Royal York	we got hotel pretty late around stoping niagar...	5	1	0.8500

	file	hotel name	review	rating	sentiment	vader
3	data\ca\155019\155495\682872848.html	Fairmont Royal York	my family relaxed stay fairmont hotel staff en...	5	1	0.9785
4	data\ca\155019\155495\683051089.html	Fairmont Royal York	we came royal york attend son wedding held sat...	1	0	-0.3493

In [0]:
```
top500_Words=getMostKwords(reviewDF['review'],500)
#print(top500_Words[:10])
def judgeExistence(topk,review,hotelDf):
  freqReview = []
  for item in review:
    #print(item)
    tempCounter = Counter()
    tempCounter.update([word for word in re.findall(r'\w+', item)])
    topkinReview = [1 if tempCounter[word] > 0 else 0 for word in topk]
    freqReview.append(topkinReview)
  freqReviewDf = pandas.DataFrame(freqReview)
  dfName = []
  for c in topk:
    dfName.append(c)
  #print(dfName[:10])
  freqReviewDf.columns = dfName
  finaldf = hotelDf[['hotel name','rating','sentiment','review']].join(
freqReviewDf)
  return finaldf
```

In [0]:
```
finaldf=judgeExistence(top500_Words,reviewDF['review'],reviewDF)
```

In [130]:
```
finaldf.head()
```

Out[130]:

	hotel name	rating	sentiment	review	i	hotel	the	room	staff	stay	great	we	breakf
0	Fairmont Royal York	5	1	i celebrate daughters birthday they wonderful ...	1	1	0	0	1	0	1	0	0
1	Fairmont Royal York	3	0	very central location sizeable hotel bit old f...	0	1	0	0	1	0	0	0	0
2	Fairmont Royal York	5	1	we got hotel pretty late around stoping niagar...	1	1	1	1	1	0	0	1	0
3	Fairmont Royal York	5	1	my family relaxed stay fairmont hotel staff en...	1	1	0	1	1	1	0	0	0

	hotel name	rating	sentiment	review	i	hotel	the	room	staff	stay	great	we	breakf
4	Fairmont Royal York	1	0	we came royal york attend son wedding held sat...	1	1	0	0	1	0	0	1	0

5 rows × 504 columns

In [0]:
```
# get Top K mutual information terms from the dataframe, X is word Y is
 label(groundtruth) distance measure, either way is okay
def getMI(k,df,topk):
    miScore = []
    for term in topk:
      miScore.append([term]+[metrics.mutual_info_score(df['sentiment'],
 df[term])])

    miScoredf = pandas.DataFrame(miScore).sort_values(1,ascending=0)
    miScoredf.columns = ['Word','MI Score']
    miScoredf=miScoredf[:k]
    return miScoredf
```

In [0]:
```
topkMI=getMI(50,finaldf,top500_Words)
```

In [133]:
```
topkMI
```

Out[133]:

	Word	MI Score
118	told	0.025272
6	great	0.020263

	Word	MI Score
142	said	0.017522
218	called	0.014254
141	asked	0.013013
213	when	0.011402
22	night	0.011133
272	call	0.010959
126	however	0.010632
21	friendly	0.010272
121	never	0.010080
195	not	0.009752
224	no	0.009427
53	definitely	0.009414
164	another	0.009247
43	excellent	0.009093
350	someone	0.008752
436	paid	0.008381
39	helpful	0.008292
205	sleep	0.008251
30	get	0.008125
206	left	0.007981
18	one	0.007850
35	like	0.007615

	Word	MI Score
3	room	0.007605
9	clean	0.007478
176	manager	0.007436
82	amazing	0.007193
204	nothing	0.007170
97	perfect	0.007149
300	bad	0.007122
4	staff	0.006974
128	wonderful	0.006966
145	door	0.006779
14	toronto	0.006561
346	pay	0.006516
364	toilet	0.006443
210	know	0.006417
45	even	0.006232
162	loved	0.006137
401	disappointed	0.006096
434	phone	0.006089
283	person	0.006058
339	ok	0.006028
471	least	0.005990
151	going	0.005812

	Word	MI Score
32	could	0.005808
23	desk	0.005778
62	spacious	0.005747
102	booked	0.005718

The top ranked words by MI are mostly verbs and adj and nouns and the adjs are more sentimentive than most of the adjs.

(b)

In [0]:
```
top500_NP=topkNP_Extraction(reviewDF['review'],500)
np500_list=[]
for item in top500_NP:
  np500_list.append(item[0])
```

In [0]:
```
def np_JudgeExistence(topk,review,hotelDf):
  freqReview = []
  grammar = "NP: {<DT>?<JJ>*<NN>}"
  parser = nltk.RegexpParser(grammar)
  tokenizer=RegexpTokenizer(r'\w+')

  for item in review:
    #print(item)
    words = tokenizer.tokenize(item)
    pos_words=nltk.pos_tag(words)
    result = parser.parse(pos_words)
    noun_phrases_list =[]
    for subtree in result.subtrees(filter = lambda t: t.label()=='NP'):
      npString=[]
      if len(subtree.leaves())==1:
        continue
      for leaf in subtree.leaves():
```

```
        npString.append(leaf[0])
      npWord=' '.join(npString)
      noun_phrases_list.append(npWord)


    tempCounter = Counter()
    tempCounter.update([word for word in noun_phrases_list])
    topkinReview = [1 if tempCounter[word] > 0 else 0 for word in topk]
    freqReview.append(topkinReview)
  freqReviewDf = pandas.DataFrame(freqReview)
  dfName = []
  for c in topk:
    dfName.append(c)
  #print(dfName[:10])
  freqReviewDf.columns = dfName
  finaldf = hotelDf[['hotel name','rating','sentiment','review']].join(
freqReviewDf)
  return finaldf
```

In [0]:
```
np_FinalDF=np_JudgeExistence(np500_list,reviewDF['review'],reviewDF)
```

In [137]:
```
np_FinalDF.head(10)
```

Out[137]:

	hotel name	rating	sentiment	review	the hotel	the room	the staff	this hotel	great location	front desk	tl locatic
0	Fairmont Royal York	5	1	i celebrate daughters birthday they wonderful ...	0	0	0	0	0	0	0

	hotel name	rating	sentiment	review	the hotel	the room	the staff	this hotel	great location	front desk	tl locatic
1	Fairmont Royal York	3	0	very central location sizeable hotel bit old f...	0	0	0	0	0	0	0
2	Fairmont Royal York	5	1	we got hotel pretty late around stoping niagar...	0	0	0	0	0	0	0
3	Fairmont Royal York	5	1	my family relaxed stay fairmont hotel staff en...	0	0	0	0	0	0	0
4	Fairmont Royal York	1	0	we came royal york attend son wedding held sat...	0	0	0	0	0	0	0
5	Fairmont Royal York	4	1	we stayed friends first part canadian holiday ...	0	1	0	0	0	0	0

	hotel name	rating	sentiment	review	the hotel	the room	the staff	this hotel	great location	front desk	tl locatic
6	Fairmont Royal York	5	1	stayed nights hotel clean room level huge staf...	0	0	0	0	0	0	0
7	Fairmont Royal York	2	0	this review culmination two experiences fairmo...	0	0	0	0	0	0	0
8	Fairmont Royal York	4	1	i neutral updates the lounge middle lobby inte...	0	0	0	0	0	0	0
9	Fairmont Royal York	4	1	great location harbour front cn corner room fl...	0	0	0	0	1	0	0

10 rows × 504 columns

In [0]:
```
topk_NPMI=getMI(50,np_FinalDF,np500_list)
```

In [139]:
```
topk_NPMI
```

Out[139]:

	Word	MI Score
5	front desk	0.006010
137	speak manager	0.004571
64	another hotel	0.003359
246	hard time	0.002678
42	another room	0.002670
220	no one	0.002614
13	great hotel	0.002508
47	next day	0.001901
44	excellent location	0.001666
213	the next day	0.001560
284	the toilet	0.001497
444	next room	0.001454
20	great service	0.001400
117	general manager	0.001329
28	great place	0.001323
140	last night	0.001204
38	all staff	0.001178
392	different hotel	0.001177
382	bad i	0.001177
23	top notch	0.001116
126	good thing	0.001092
92	great job	0.001029

	Word	MI Score
8	friendly staff	0.001020
179	second night	0.001011
93	great value	0.000990
46	perfect location	0.000966
48	great stay	0.000966
95	queen street	0.000950
32	excellent service	0.000929
228	the tv	0.000916
465	average hotel	0.000909
496	the noise	0.000909
498	middle night	0.000909
53	recommend hotel	0.000897
33	short walk	0.000834
116	great i	0.000831
301	the night	0.000823
40	next door	0.000769
2	the staff	0.000745
416	the door	0.000734
365	location hotel	0.000734
427	bad experience	0.000734
378	cleaning staff	0.000734
383	double bed	0.000734

	Word	MI Score
282	i hotel	0.000698
29	canada wonderland	0.000680
30	easy access	0.000680
153	wonderful hotel	0.000672
151	next trip	0.000672
69	great time	0.000659

From the noun phrases list we can see the majority of the them are terms that the customers care about the most.

Q4

(a)

In [0]:
```
# Simple example of getting pairwise mutual information of a term
def pmiCal(df, x):
    pmilist=[]
    for i in [1,0]:
        for j in [0,1]:
            px = sum(df['sentiment']==i)/len(df)
            py = sum(df[x]==j)/len(df)
            pxy = len(df[(df['sentiment']==i) & (df[x]==j)])/len(df)
            if pxy==0:#Log 0 cannot happen
                pmi = math.log((pxy+0.0001)/(px*py))
            else:
                pmi = math.log(pxy/(px*py))
            pmilist.append([i]+[j]+[px]+[py]+[pxy]+[pmi])
    pmidf = pandas.DataFrame(pmilist)
```

```
    pmidf.columns = ['x','y','px','py','pxy','pmi']
    return pmidf
```

In [0]:
```
def pmiIndivCal(df,x,gt, label_column='sentiment'):
    px = sum(df[label_column]==gt)/len(df)
    py = sum(df[x]==1)/len(df)
    pxy = len(df[(df[label_column]==gt) & (df[x]==1)])/len(df)
    if pxy==0:#Log 0 cannot happen
        pmi = math.log((pxy+0.0001)/(px*py))
    else:
        pmi = math.log(pxy/(px*py))
    return pmi
```

In [0]:
```
def pmiForAllCal(df, topk):
    #Try calculate all the pmi for top k and store them into one pmidf
 dataframe
    pmilist = []
    pmiposlist = []
    pmineglist = []
    for word in tqdm(topk):
        pmilist.append([word]+[pmiCal(df,word)])
        pmiposlist.append([word]+[pmiIndivCal(df,word,1,'sentiment')])
        pmineglist.append([word]+[pmiIndivCal(df,word,0,'sentiment')])
    pmidf = pandas.DataFrame(pmilist)
    pmiposlist = pandas.DataFrame(pmiposlist)
    pmineglist = pandas.DataFrame(pmineglist)
    pmiposlist.columns = ['word','pmi']
    pmineglist.columns = ['word','pmi']
    pmidf.columns = ['word','pmi']
    return pmiposlist, pmineglist, pmidf
```

In [143]:
```
# Compute PMI for all terms and all possible labels
pmiposlist, pmineglist, pmidf=pmiForAllCal(finaldf, top500_Words)
```

```
100%|██████████| 500/500 [00:18<00:00, 27.43it/s]
```

In [144]:
```
pmiposlist.sort_values('pmi',ascending=0).head(50)
```

Out[144]:

	word	pmi
477	david	0.179214
162	loved	0.170815
128	wonderful	0.169065
187	highly	0.165865
308	accommodating	0.163254
189	fantastic	0.162949
391	thanks	0.157085
97	perfect	0.156037
133	enjoyed	0.155901
82	amazing	0.153415
464	aquarium	0.151419
130	beautiful	0.149700
340	awesome	0.148797
322	efficient	0.147516
53	definitely	0.145690
358	delicious	0.142152
43	excellent	0.141334
373	beyond	0.137647
279	attractions	0.129103
112	cn	0.128858
327	welcoming	0.128570
485	entertainment	0.126644

	word	pmi
62	spacious	0.123708
39	helpful	0.123147
368	spa	0.120224
217	mall	0.119835
244	love	0.118341
243	center	0.117531
478	game	0.117531
455	from	0.117345
109	tower	0.117080
369	subway	0.116594
499	attentive	0.115744
153	centre	0.115636
387	district	0.115313
123	shopping	0.113820
85	city	0.113410
241	airport	0.110755
271	concierge	0.110659
68	restaurants	0.110647
456	drinks	0.110079
301	plenty	0.108295
215	union	0.107860
51	recommend	0.107675

	word	pmi
96	easy	0.106635
21	friendly	0.106027
221	thank	0.105625
6	great	0.105394
430	welcome	0.105165
362	views	0.102801

In [145]:
```
pmineglist.sort_values('pmi',ascending=0).head(50)
```

Out[145]:

	word	pmi
118	told	1.232782
436	paid	1.119129
272	call	1.115558
218	called	1.112911
142	said	1.112126
350	someone	1.084268
434	phone	1.034869
388	card	1.028354
452	management	1.017919
471	least	1.016201
472	loud	0.998501
213	when	0.993966
364	toilet	0.983234

	word	pmi
474	probably	0.970592
141	asked	0.961322
224	no	0.958496
346	pay	0.949711
401	disappointed	0.946829
494	tried	0.937877
339	ok	0.933496
300	bad	0.931197
453	noisy	0.918458
195	not	0.904816
351	hear	0.904569
479	waiting	0.893141
487	later	0.890870
357	hours	0.889302
206	left	0.881864
283	person	0.879238
205	sleep	0.877423
424	leave	0.870668
415	charge	0.866329
372	air	0.864970
176	manager	0.864305
164	another	0.861742

	word	pmi
406	money	0.854252
473	after	0.850865
126	however	0.850720
483	either	0.841109
210	know	0.823698
204	nothing	0.822936
317	give	0.821167
393	issues	0.817753
470	doors	0.816180
121	never	0.805597
458	cleaning	0.805597
260	open	0.802026
476	hour	0.789409
264	problem	0.789151
305	put	0.787936

PMI analysis result showed some difference from the MI, in the positive list the first ranked was a name 'David' and I looked into the dataset and found he was a staff at Toronto Inter Hotel and he had great performance at his work so many customers mentioned him in their reviews which are mostly positive, the reason perhaps is PMI focus on single pair not the entire dataset.

(b)

In [146]:
```
np_Pmiposlist, np_Pmineglist, np_Pmidf=pmiForAllCal(np_FinalDF, np500_l
```

```
ist)
```

```
100%|██████████| 500/500 [00:16<00:00, 31.04it/s]
```

In [147]:
```
np_Pmiposlist.sort_values('pmi',ascending=0).head(50)
```

Out[147]:

	word	pmi
186	beautiful room	0.203605
93	great value	0.203605
499	live music	0.203605
332	small hotel	0.203605
283	outside hotel	0.203605
286	great selection	0.203605
287	happy stay	0.203605
288	attentive staff	0.203605
292	special mention	0.203605
151	next trip	0.203605
306	clean service	0.203605
319	spacious clean	0.203605
322	excellent food	0.203605
324	prime location	0.203605
337	downtown hotel	0.203605
275	downtown area	0.203605
339	exceptional service	0.203605
342	comfortable hotel	0.203605

	word	pmi
343	beautiful view	0.203605
344	another stay	0.203605
345	entire stay	0.203605
352	fantastic hotel	0.203605
355	this great hotel	0.203605
358	the adelaide hotel	0.203605
359	delicious breakfast	0.203605
360	stay area	0.203605
277	willing help	0.203605
260	swiss chalet	0.203605
268	restaurant staff	0.203605
214	right heart	0.203605
153	wonderful hotel	0.203605
167	last week	0.203605
190	amazing staff	0.203605
191	large room	0.203605
194	fantastic i	0.203605
195	fantastic location	0.203605
200	friendly hotel	0.203605
207	front street	0.203605
208	wonderful staff	0.203605
116	great i	0.203605

	word	pmi
217	great shopping	0.203605
371	impeccable service	0.203605
238	i town	0.203605
244	excellent customer	0.203605
245	the king	0.203605
247	the valet	0.203605
248	overall great stay	0.203605
95	queen street	0.203605
255	every need	0.203605
92	great job	0.203605

In [148]:
```
np_Pmineglist.sort_values('pmi',ascending=0).head(50)
```

Out[148]:

	word	pmi
137	speak manager	1.631024
246	hard time	1.596338
444	next room	1.537498
220	no one	1.524594
392	different hotel	1.403966
382	bad i	1.403966
284	the toilet	1.373195
498	middle night	1.355176
465	average hotel	1.355176

	word	pmi
496	the noise	1.355176
213	the next day	1.323924
488	key card	1.286183
64	another hotel	1.270435
427	bad experience	1.221645
416	the door	1.221645
365	location hotel	1.221645
383	double bed	1.221645
378	cleaning staff	1.221645
301	the night	1.180823
117	general manager	1.161020
42	another room	1.161020
228	the tv	1.152652
453	slow i	1.132033
179	second night	1.132033
466	first day	1.132033
481	the person	1.132033
423	poor customer	1.132033
340	nice location	1.103862
140	last night	1.103862
282	i hotel	1.085513
126	good thing	1.049794

	word	pmi
379	the mattress	0.998501
251	the next morning	0.998501
480	the management	0.998501
303	i guess	0.998501
430	the rest	0.998501
321	first room	0.998501
399	stayed business	0.998501
436	half hour	0.998501
295	boutique hotel	0.998501
226	small room	0.998501
437	much i	0.998501
47	next day	0.973809
206	last time	0.918458
253	sofa bed	0.903191
363	separate room	0.880718
356	sure room	0.880718
165	the air	0.864970
408	spend time	0.844350
452	the window	0.844350

One interesting outcome is in noun phrases you hardly see any negative phrases in positive reviews but the positive phrases still appears in negative reviews. In addition, compared to MI, the rank of these positive phrases are lower in the neg_list compared with MI.

(c)

In [0]:

```
# Simple example of getting pairwise mutual information of a term
def single_pmiCal(df, x):
    pmilist=[]
    for i in [1,0]:
        for j in [0,1]:
            px = sum(df['sentiment']==i)/len(df)
            py = sum(df[x]==j)/len(df)
            pxy = len(df[(df['sentiment']==i) & (df[x]==j)])/len(df)
            if px==0 or py==0:
              pmi=0
              pxy=0
              pmilist.append([i]+[j]+[px]+[py]+[pxy]+[pmi])
              continue
            if pxy==0:#Log 0 cannot happen
                pmi = math.log((pxy+0.0001)/(px*py))
            else:
                pmi = math.log(pxy/(px*py))
            pmilist.append([i]+[j]+[px]+[py]+[pxy]+[pmi])
    pmidf = pandas.DataFrame(pmilist)
    pmidf.columns = ['x','y','px','py','pxy','pmi']
    return pmidf

def single_pmiIndivCal(df,x,gt, label_column='sentiment'):
    px = sum(df[label_column]==gt)/len(df)
    py = sum(df[x]==1)/len(df)
    pxy = len(df[(df[label_column]==gt) & (df[x]==1)])/len(df)
    if px==0 or py==0:
      pmi=0
      return pmi
    if pxy==0:#Log 0 cannot happen
        pmi = math.log((pxy+0.0001)/(px*py))
    else:
        pmi = math.log(pxy/(px*py))
    return pmi
```

```
def single_pmiForAllCal(df, topk):
    #Try calculate all the pmi for top k and store them into one pmidf
 dataframe
    pmilist = []
    pmiposlist = []
    pmineglist = []
    for word in tqdm(topk):
        pmilist.append([word]+[single_pmiCal(df,word)])
        pmiposlist.append([word]+[single_pmiIndivCal(df,word,1,'sentiment')])
        pmineglist.append([word]+[single_pmiIndivCal(df,word,0,'sentiment')])
    pmidf = pandas.DataFrame(pmilist)
    pmiposlist = pandas.DataFrame(pmiposlist)
    pmineglist = pandas.DataFrame(pmineglist)
    pmiposlist.columns = ['word','pmi']
    pmineglist.columns = ['word','pmi']
    pmidf.columns = ['word','pmi']
    return pmiposlist, pmineglist, pmidf
```

In [0]:
```
shangrila_Review=finaldf[finaldf['hotel name']=='Shangri-La Hotel Toronto']
super8_Review=finaldf[finaldf['hotel name']=='Super 8 by Wyndham Toronto North']
```

In [151]:
```
shangrila_Review.head()
```

Out[151]:

	hotel name	rating	sentiment	review	i	hotel	the	room	staff	stay	great	we	bı
2024	Shangri-La Hotel Toronto	5	1	excellent service helpful friendly really enjo...	1	0	0	0	0	0	0	0	0

	hotel name	rating	sentiment	review	i	hotel	the	room	staff	stay	great	we	bı
2025	Shangri-La Hotel Toronto	5	1	i stayed many five star hotels numerous cities...	1	1	0	0	0	0	1	0	0
2026	Shangri-La Hotel Toronto	2	0	got married restaurant across street decided s...	1	1	1	1	0	1	1	1	0
2027	Shangri-La Hotel Toronto	5	1	pleasure stay toronto service decor amazing al...	0	0	0	0	0	1	0	0	0
2028	Shangri-La Hotel Toronto	5	1	outstanding service added touches beautiful sp...	0	0	1	0	0	1	0	0	0

5 rows × 504 columns

In [152]:
```
shangrila_pmiposlist, shangrila_pmineglist, shangrila_pmidf=single_pmiForAllCal(shangrila_Review, top500_Words)
```

```
100%|██████████| 500/500 [00:08<00:00, 59.75it/s]
```

In [153]:
```
shangrila_pmiposlist.sort_values('pmi',ascending=0).head(50)
```

Out[153]:

	word	pmi
499	attentive	0.05657
373	beyond	0.05657
275	needs	0.05657
150	tv	0.05657
82	amazing	0.05657
81	business	0.05657
80	view	0.05657
268	gym	0.05657
267	side	0.05657
384	tea	0.05657
263	something	0.05657
74	beds	0.05657
387	district	0.05657
390	town	0.05657
391	thanks	0.05657
392	years	0.05657
156	feel	0.05657
398	visiting	0.05657
404	enjoy	0.05657

	word	pmi
319	ca	0.05657
406	money	0.05657
162	loved	0.05657
252	help	0.05657
165	times	0.05657
414	member	0.05657
148	find	0.05657
371	selection	0.05657
53	definitely	0.05657
370	hard	0.05657
322	efficient	0.05657
308	accommodating	0.05657
129	lots	0.05657
301	plenty	0.05657
335	worth	0.05657
116	quiet	0.05657
115	big	0.05657
113	work	0.05657
294	club	0.05657
341	in	0.05657
293	offered	0.05657
108	price	0.05657

	word	pmi
343	impressed	0.05657
344	given	0.05657
136	a	0.05657
137	kids	0.05657
103	modern	0.05657
139	found	0.05657
287	far	0.05657
358	delicious	0.05657
359	plus	0.05657

In [154]:
```
shangrila_pmineglist.sort_values('pmi',ascending=0).head(50)
```

Out[154]:

	word	pmi
314	ritz	2.900422
483	either	2.900422
469	internet	2.900422
443	point	2.900422
393	issues	2.389596
476	hour	2.207275
300	bad	2.207275
303	towels	2.207275
312	housekeeping	2.207275
313	deal	2.207275

	word	pmi
439	three	2.207275
196	across	2.207275
349	kept	2.207275
351	hear	2.207275
388	card	2.207275
394	done	2.207275
395	items	2.207275
419	let	2.207275
292	working	2.207275
227	happy	1.984131
86	hot	1.984131
452	management	1.801810
199	near	1.801810
446	floors	1.801810
260	open	1.801810
264	problem	1.801810
444	reasonable	1.801810
265	late	1.801810
195	not	1.801810
210	know	1.801810
158	needed	1.801810
304	cold	1.801810

	word	pmi
216	days	1.801810
120	use	1.801810
471	least	1.801810
338	light	1.801810
425	may	1.801810
377	fast	1.801810
389	due	1.801810
423	sofa	1.801810
220	outside	1.801810
87	got	1.696449
119	street	1.647659
245	used	1.647659
222	last	1.647659
401	disappointed	1.647659
424	leave	1.647659
270	look	1.647659
332	end	1.647659
218	called	1.514128

In [155]:
```
super_pmiposlist, super_pmineglist, super_pmidf=single_pmiForAllCal(super8_Review, top500_Words)
```

```
100%|██████████| 500/500 [00:08<00:00, 61.20it/s]
```

In [156]:
```
super_pmiposlist.sort_values('pmi',ascending=0).head(50)
```

Out[156]:

	word	pmi
322	efficient	0.634878
391	thanks	0.634878
383	travel	0.634878
358	delicious	0.634878
326	marriott	0.634878
70	bar	0.634878
311	wife	0.634878
281	but	0.634878
279	attractions	0.634878
258	short	0.634878
243	center	0.634878
229	elevator	0.634878
189	fantastic	0.634878
186	lounge	0.634878
152	kitchen	0.634878
88	walking	0.634878
85	city	0.634878
386	corner	0.634878
80	view	0.634878
399	expect	0.634878
443	point	0.634878

	word	pmi
422	expensive	0.634878
490	forward	0.634878
485	entertainment	0.634878
460	menu	0.634878
450	polite	0.634878
427	equipped	0.634878
433	brought	0.634878
411	appointed	0.634878
455	from	0.634878
468	greeted	0.634878
435	quickly	0.634878
437	areas	0.634878
130	beautiful	0.634878
43	excellent	0.634878
117	distance	0.634878
103	modern	0.634878
48	restaurant	0.634878
187	highly	0.634878
257	options	0.634878
454	keep	0.634878
277	complimentary	0.634878
287	far	0.634878

	word	pmi
301	plenty	0.634878
308	accommodating	0.634878
324	getting	0.634878
426	highway	0.634878
340	awesome	0.634878
423	sofa	0.634878
499	attentive	0.634878

In [157]:
```
super_pmineglist.sort_values('pmi',ascending=0).head(50)
```

Out[157]:

	word	pmi
472	loud	0.755023
346	pay	0.755023
349	kept	0.755023
350	someone	0.755023
351	hear	0.755023
372	air	0.755023
253	you	0.755023
223	wedding	0.755023
491	past	0.755023
378	elevators	0.755023
381	dining	0.755023

	word	pmi
390	town	0.755023
119	street	0.755023
498	line	0.755023
126	however	0.755023
282	and	0.755023
280	provided	0.755023
374	hockey	0.755023
364	toilet	0.755023
473	after	0.755023
365	several	0.755023
403	living	0.755023
492	makes	0.755023
315	fine	0.755023
465	safe	0.755023
321	part	0.755023
328	main	0.755023
353	as	0.755023
345	different	0.755023
330	level	0.755023
333	train	0.755023
276	inn	0.755023
342	hall	0.755023

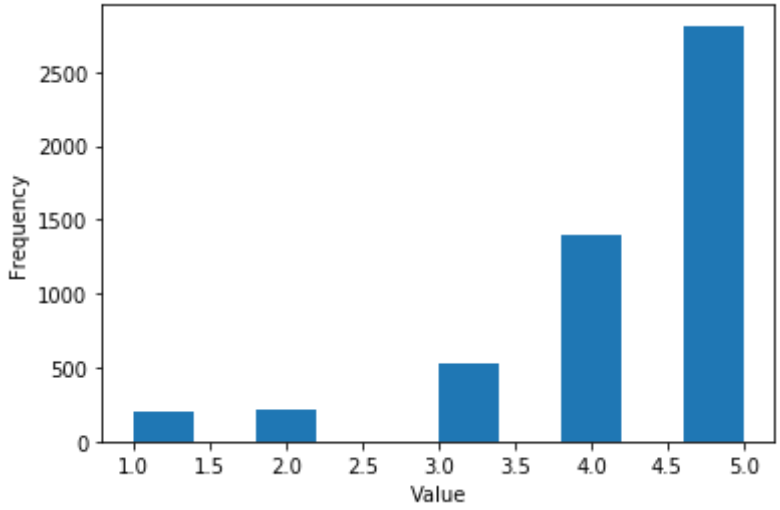
	word	pmi
274	professional	0.755023
414	member	0.755023
247	husband	0.755023
486	residence	0.755023
431	standard	0.755023
479	waiting	0.755023
452	management	0.755023
239	he	0.755023
478	game	0.755023
483	either	0.755023
438	upgraded	0.755023
234	reception	0.755023
442	actually	0.755023
453	noisy	0.755023
209	dinner	0.755023
428	less	0.755023
487	later	0.755023

From the list we can tell that shangeri la has merits of good big custom room and modern equipments but some customers may complained about the internet and the temperature. For super 8 hotel, it benefits are well entertainments and club easy to travel, but it is also complained about noise.

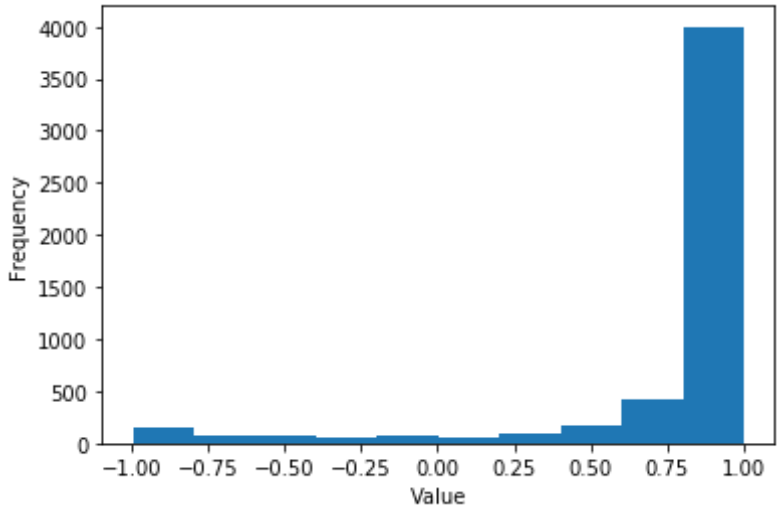
#Q5

(a)Histogram

In [158]:
```
plt.hist(reviewDF['rating'].values)

plt.xlabel("Value")
plt.ylabel("Frequency")
fig = plt.gcf()
```

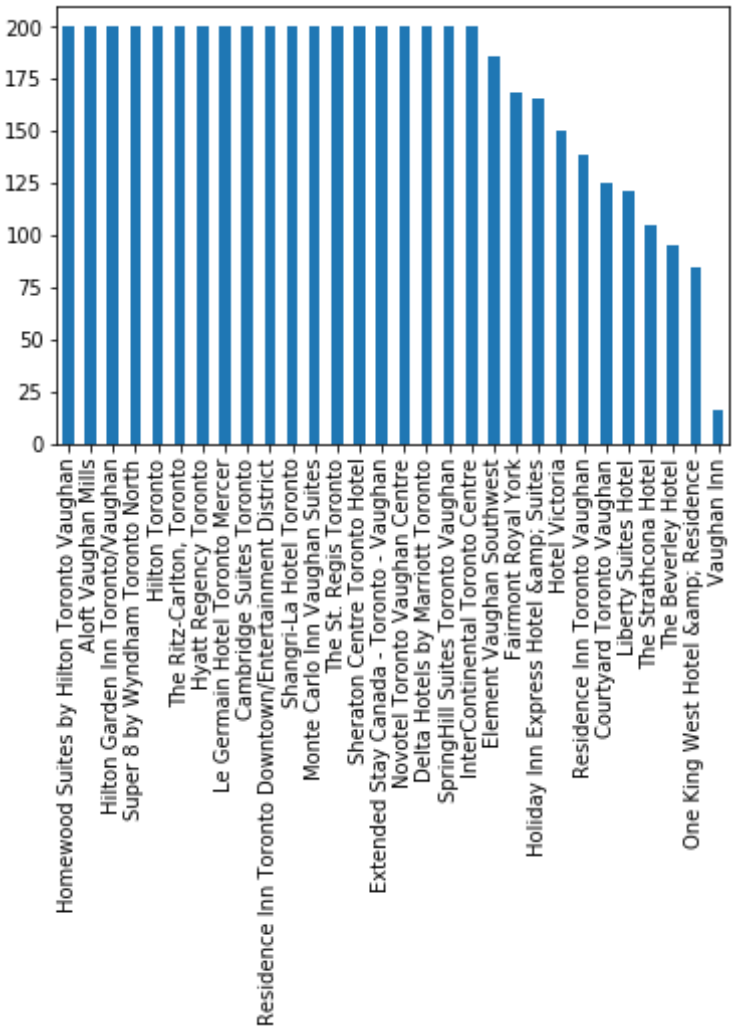


In [159]:
```
plt.hist(vaderDF['vader'].values)

plt.xlabel("Value")
plt.ylabel("Frequency")
fig = plt.gcf()
```



The distribution of Vader prediction is more concentrated around 0.75 to 1 than ground truth.

In [160]:
```
rates=reviewDF['hotel name'].value_counts().plot(kind = 'bar')
```



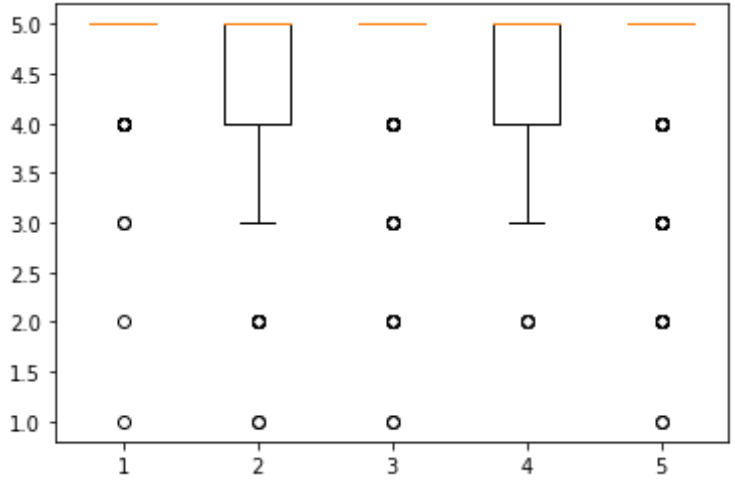
Since I implemented a constrain on the max review number retrived from a single hotel, the max review number for a single hotel is 200. From the histogram we can see that the hotels that are close to Toronto has more reviews than hotels that are far away from Toronto and it is within my expectation.

(b)Boxplots

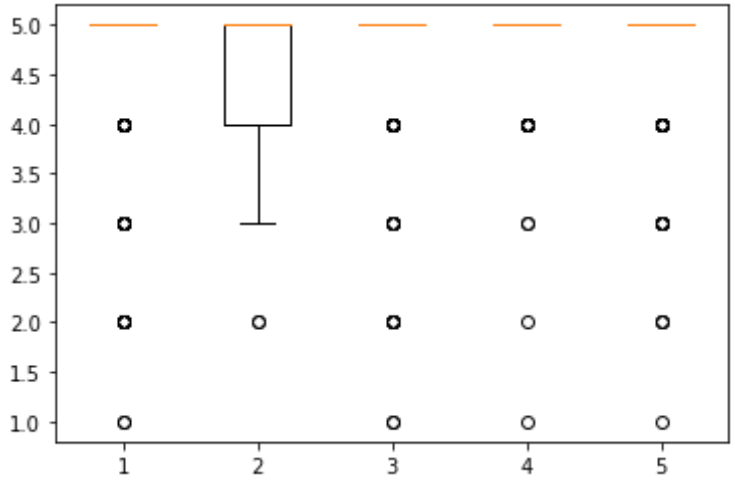
In [0]:
```
import seaborn as sns
vaderHotel_Name,values=zip(*sorted_meanVader)
top5Vader=vaderHotel_Name[-5:]

trueHotel_Name,truevalues=zip(*sorted_meanTruth)
top5True=trueHotel_Name[-5:]
```

In [0]:
```
vader_Data=[]
true_Data=[]
for hotel in top5Vader:
  hotelData=reviewDF.loc[reviewDF['hotel name']==hotel,'rating']

  vader_Data.append(hotelData)

for hotel in top5True:
  hotelData=reviewDF.loc[reviewDF['hotel name']==hotel,'rating']
  true_Data.append(hotelData)
```

In [163]:
```
plt.figure()
plt.boxplot(vader_Data)
plt.show()
```



In [164]:
```
plt.figure()
plt.boxplot(true_Data)
plt.show()
```



In [165]:
```
top5True
```

Out[165]:
```
('The St. Regis Toronto',
 'Le Germain Hotel Toronto Mercer',
 'The Ritz-Carlton, Toronto',
 'One King West Hotel &amp;amp; Residence',
 'Shangri-La Hotel Toronto')
```

In [166]:
```
variance_true=[]
variance_vader=[]
mean_true=[]
mean_vader=[]
for item in true_Data:
  ar=item.values
  v=np.var(ar)
  m=np.mean(ar)
  variance_true.append(v)
  mean_true.append(m)
```

```
for item in vader_Data:
  ar=item.values
  v=np.var(ar)
  m=np.mean(ar)
  variance_vader.append(v)
  mean_vader.append(m)
print(variance_vader)
print(variance_true)
print(mean_vader)
print(mean_true)
```

```
[0.4711418685121109, 0.759375, 0.5739, 0.522775, 0.6903999999999999]
[0.6903999999999999, 0.4003999999999999, 0.5739, 0.4711418685121109, 0.
40997500000000003]
[4.694117647058824, 4.475, 4.69, 4.585, 4.64]
[4.64, 4.64, 4.69, 4.694117647058824, 4.745]
```

From my perspective, the mean and variance is better than boxplot because it is more precise and makes the compare easier, but the boxplot shows the trend more straight forward.

Hotel Name	Vader-Mean	Vader-Variance	Ground Truth-Mean	Ground Truth-Variance
The St. Regis Toronto	4.694117647058824	0.4711418685121109	4.64	0.6903999999999999
Le Germain Hotel Toronto Mercer	4.475	0.759375	4.64	0.4003999999999999
The Ritz-Carlton, Toronto	4.69	0.5739	4.69	0.5739

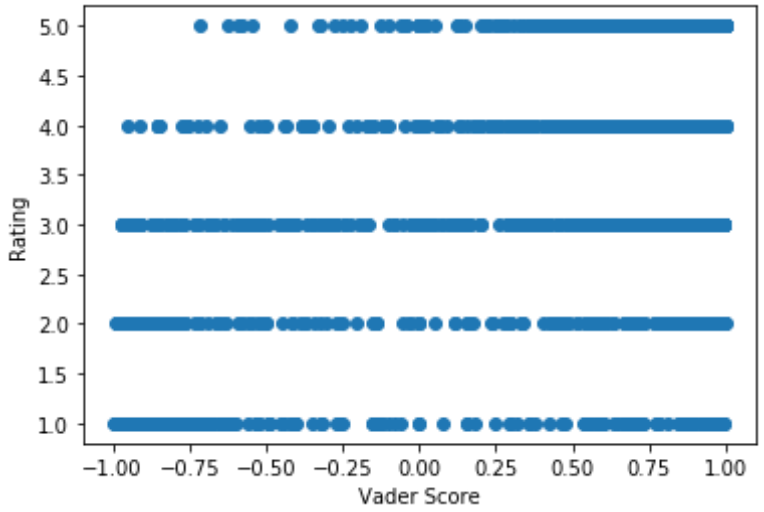
Hotel Name	Vader-Mean	Vader-Variance	Ground Truth-Mean	Ground Truth-Variance
One King West Hotel & Residence	4.585	0.522775	4.694117647058824	0.4711418685121109
Shangri-La Hotel Toronto	4.64	0.6903999999999999	4.745	0.40997500000000003

(c)Scatterplots and heatmaps

(a)

In [167]:
```
y = reviewDF['rating'].values
x = reviewDF['vader'].values
plt.plot(x,y,"o")
plt.ylabel('Rating')
plt.xlabel('Vader Score')
```

Out[167]:
```
Text(0.5, 0, 'Vader Score')
```



The nodes are concentrated on both ends,which means the rating and predictions are tends to be extreme.

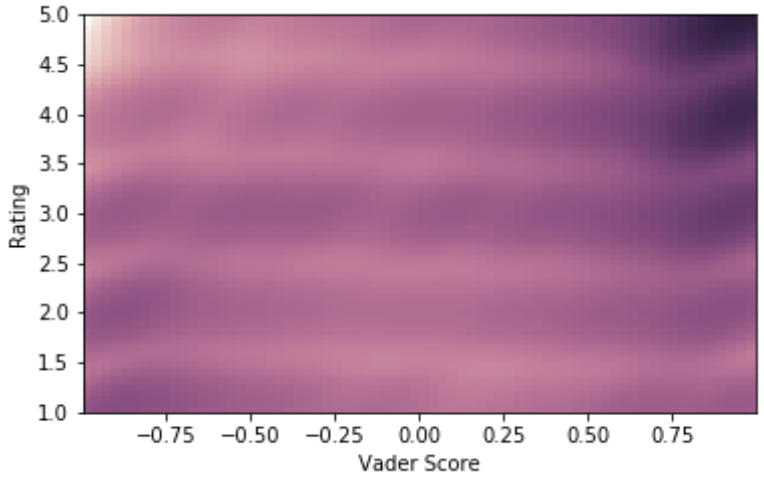
Star ratings vs. Vader tells me the relation between the rating and prediction. We can see that the left upper corner which means rating is high but predicted as negative is sparse means the Vader rarely predict a very positive review as totally negative but the right lower corner is dense which means Vader sometimes predict the negative reviews as the positive ones.

In [0]:
```
from scipy.stats.kde import gaussian_kde

k = gaussian_kde(np.vstack([x, y]))
xi, yi = np.mgrid[x.min():x.max():x.size**0.5*1j,y.min():y.max():y.size
**0.5*1j]
zi = k(np.vstack([xi.flatten(), yi.flatten()]))
```

In [169]:
```
cmap = sns.cubehelix_palette(light=1, as_cmap=True)
fig = plt.figure(figsize=(6,8))
ax1 = fig.add_subplot(211)

ax1.pcolormesh(xi, yi, np.log10(zi.reshape(xi.shape)), cmap=cmap)
```

```
ax1.set_xlim(x.min(), x.max())
ax1.set_ylim(y.min(), y.max())

ax1.set_xlabel('Vader Score')
ax1.set_ylabel('Rating')
```

Out[169]: Text(0, 0.5, 'Rating')



The trend is similar to the scatter plot with both dense ends and sparse in left upper corner.

(b)

In [0]:
```
length_List=[]
for item in reviewDF['review'].values:
  length=len(item)
  length_List.append(length)
reviewDF['length']=length_List
```

In [171]:
```
reviewDF.head()
```

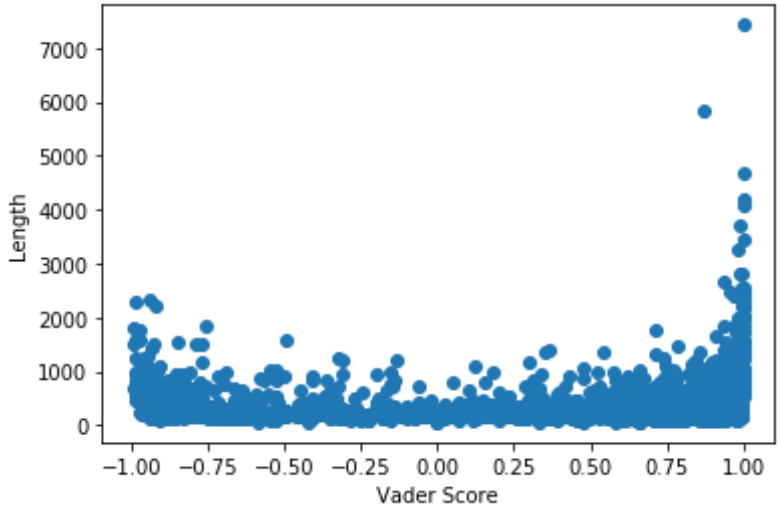
Out[171]:

	file	hotel name	review	rating	sentiment	vader	le
0	data\ca\155019\155495\682627838.html	Fairmont Royal York	i celebrate daughters birthday they wonderful ...	5	1	0.9818	1
1	data\ca\155019\155495\682656607.html	Fairmont Royal York	very central location sizeable hotel bit old f...	3	0	0.8370	1
2	data\ca\155019\155495\682676167.html	Fairmont Royal York	we got hotel pretty late around stoping niagar...	5	1	0.8500	1
3	data\ca\155019\155495\682872848.html	Fairmont Royal York	my family relaxed stay fairmont hotel staff en...	5	1	0.9785	1

	file	hotel name	review	rating	sentiment	vader	le
4	data\ca\155019\155495\683051089.html	Fairmont Royal York	we came royal york attend son wedding held sat...	1	0	-0.3493	6

In [172]:
```
y = reviewDF['length'].values
x = reviewDF['vader'].values
plt.plot(x, y,"o")
plt.ylabel('Length')
plt.xlabel('Vader Score')
```

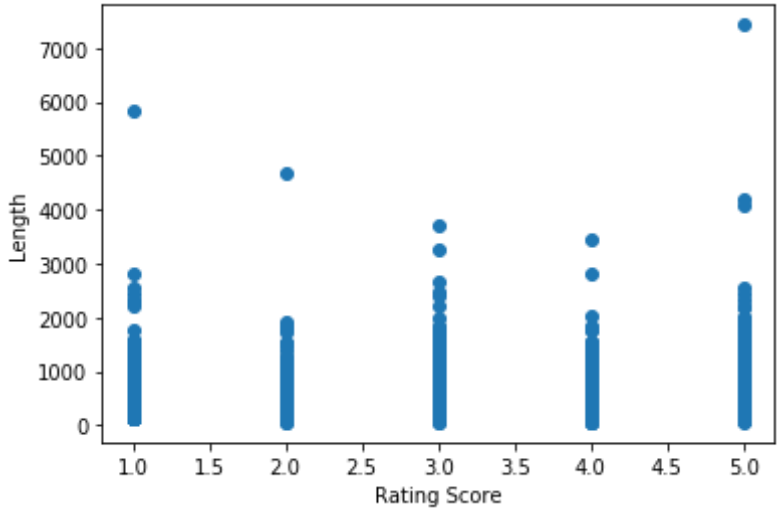
Out[172]: Text(0.5, 0, 'Vader Score')



In [173]:
```
y = reviewDF['length'].values
x = reviewDF['rating'].values
```

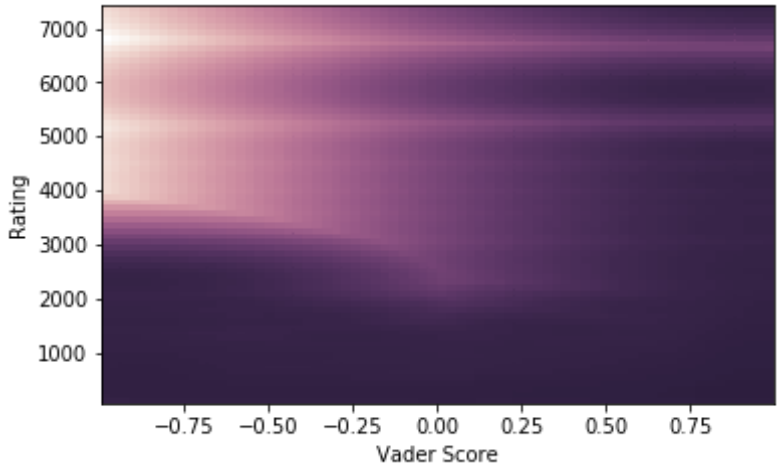
```
plt.plot(x, y,"o")
plt.ylabel('Length')
plt.xlabel('Rating Score')
```

Out[173]: Text(0.5, 0, 'Rating Score')



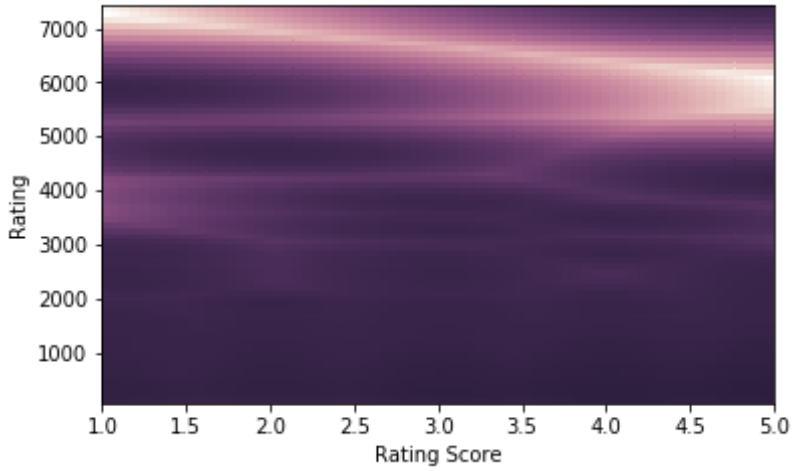
In [174]:
```
y = reviewDF['length'].values
x = reviewDF['vader'].values
k = gaussian_kde(np.vstack([x, y]))
xi, yi = np.mgrid[x.min():x.max():x.size**0.5*1j,y.min():y.max():y.size
**0.5*1j]
zi = k(np.vstack([xi.flatten(), yi.flatten()]))
cmap = sns.cubehelix_palette(light=1, as_cmap=True)
fig = plt.figure(figsize=(6,8))
ax1 = fig.add_subplot(211)

ax1.pcolormesh(xi, yi, np.log10(zi.reshape(xi.shape)), cmap=cmap)

ax1.set_xlim(x.min(), x.max())
ax1.set_ylim(y.min(), y.max())

ax1.set_xlabel('Vader Score')
ax1.set_ylabel('Rating')
```

Out[174]: Text(0, 0.5, 'Rating')



In [175]:
```
y = reviewDF['length'].values
x = reviewDF['rating'].values
k = gaussian_kde(np.vstack([x, y]))
xi, yi = np.mgrid[x.min():x.max():x.size**0.5*1j,y.min():y.max():y.size
**0.5*1j]
zi = k(np.vstack([xi.flatten(), yi.flatten()]))
cmap = sns.cubehelix_palette(light=1, as_cmap=True)
fig = plt.figure(figsize=(6,8))
ax1 = fig.add_subplot(211)

ax1.pcolormesh(xi, yi, np.log10(zi.reshape(xi.shape)), cmap=cmap)

ax1.set_xlim(x.min(), x.max())
ax1.set_ylim(y.min(), y.max())

ax1.set_xlabel('Rating Score')
ax1.set_ylabel('Rating')
```

Out[175]: Text(0, 0.5, 'Rating')



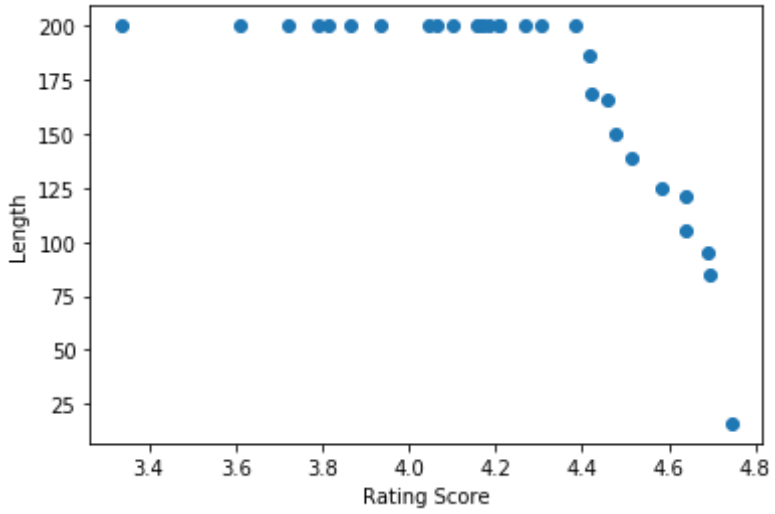
Highly positive reviews and highly negative reviews tends to have more length and the neutral reviews (like3) has the sortest length of reviews.

(c)

In [0]:
```
rateNum=reviewDF['hotel name'].value_counts()
mean_VaderList=[]
for item in sorted_meanVader:
  mean_VaderList.append(item[1])
mean_TrueList=[]
for item in sorted_meanTruth:
  mean_TrueList.append(item[1])
```

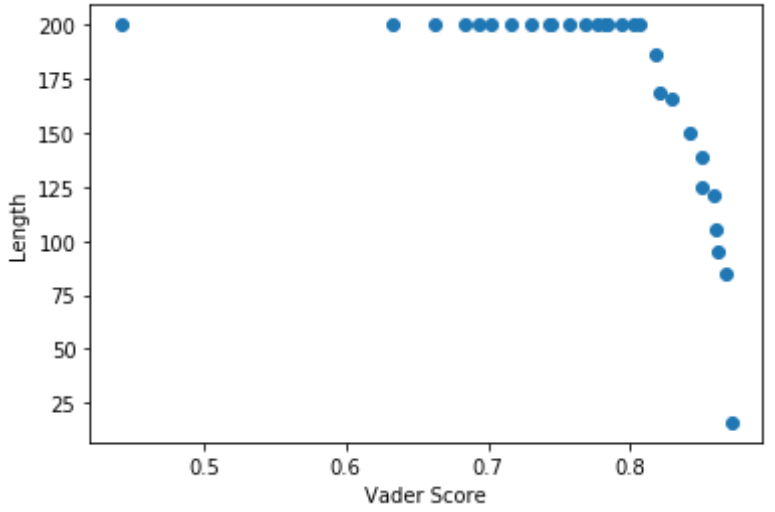
In [177]:
```
y = rateNum.values
x = mean_TrueList
plt.plot(x, y,"o")
plt.ylabel('Length')
plt.xlabel('Rating Score')
```

Out[177]:
```
Text(0.5, 0, 'Rating Score')
```



In [178]:
```
y = rateNum.values
x = mean_VaderList
plt.plot(x, y,"o")
plt.ylabel('Length')
plt.xlabel('Vader Score')
```

Out[178]: Text(0.5, 0, 'Vader Score')



Vader prediction is more concentrated than the true rating distribution.