# Sentiment Analysis on 515k Hotel Reviews

Xin Qu (xinq2@illinois.edu), Biruo Zhao (biruoz2@illinois.edu)

## Background

Tourism industry has transitioned from a brick-and-mortar and person-to-person business into a digital savvy and omnipresent travel service network. Taking booking.com for example, as one of the most successful Online Travel Agency (OTA), its website and mobile apps are available in over 40 languages, offer 28,988,780 total reported listings, and cover 144,354 destinations in 229 countries and territories worldwide. Every day, more than 1,550,000 room nights are reserved on this platform. Therefore, how to retrieve, analyze, and categorize those emotional and experiential elements of tourist activities and capitalize on those digital footprints has become a great challenge and major concern of tourism businesses. Sentiment Analysis comes into the rescue, Sentiment Analysis basically refers to the use of computational linguistics and natural language processing to analyze text and identify its subjective information (Alaei, Becken, & Stantic, 2017). Opinion Mining or Sentiment Analysis is based on the idea of unlocking the hidden value of opinions to achieve deeper understanding of customers' need and more informed and actional business insights.

## About the Data

This dataset was scraped from Booking.com by Jason Liu and publicly available on Kaggle (https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe).

The aim of applying Sentiment Analysis to this dataset is two-fold:

### *1. for hotel managers:*

Hotels can leverage opinion polarity and sentiment topic recognition to gain a deeper understanding of customers' feedback and their drivers. Also, to extract customers' emotional tone from the reviews they posted will help provide good opportunities for hotels to re-evaluate and improve their customer service and products accordingly.

### *2. for potential customers:*

Sentiment Analysis helps answer questions like if two hotels have the same review score, how can a potential customer find the right hotel?

All customers should no longer be placed in the one bucket but demand different service and products to satisfy certain needs. Some customers have high standards on hotel staff that take care of them and create guest experience. Some guests want spaciousness and quality of facilities to achieve a restful sleep. For some guests, price or good value may be the most key factor in their decision to book. And others are willing to pay extra to indulge themselves with high-end experience in their travel. In attempt to drive right customers to right hotels, we need to find a way to evaluate and sort hotels by different features.

To utilize Sentiment Analysis to its full potential, we bring a trending concept "Aspect-based Sentiment Analysis" into this project. Different from a traditional sentiment classification task which tends to treat an entire entity as a whole, Aspect-based Sentiment Analysis focuses on how to estimate sentiment score for different aspects of an entity, how positive or negative the opinions are on average for each aspect.

## Initial Observation (data_process.ipynb)

This dataset contains 515,738 customer reviews and score of 1492 luxury hotels across Europe gathered by Bookings.com from August 2015 to August 2017.

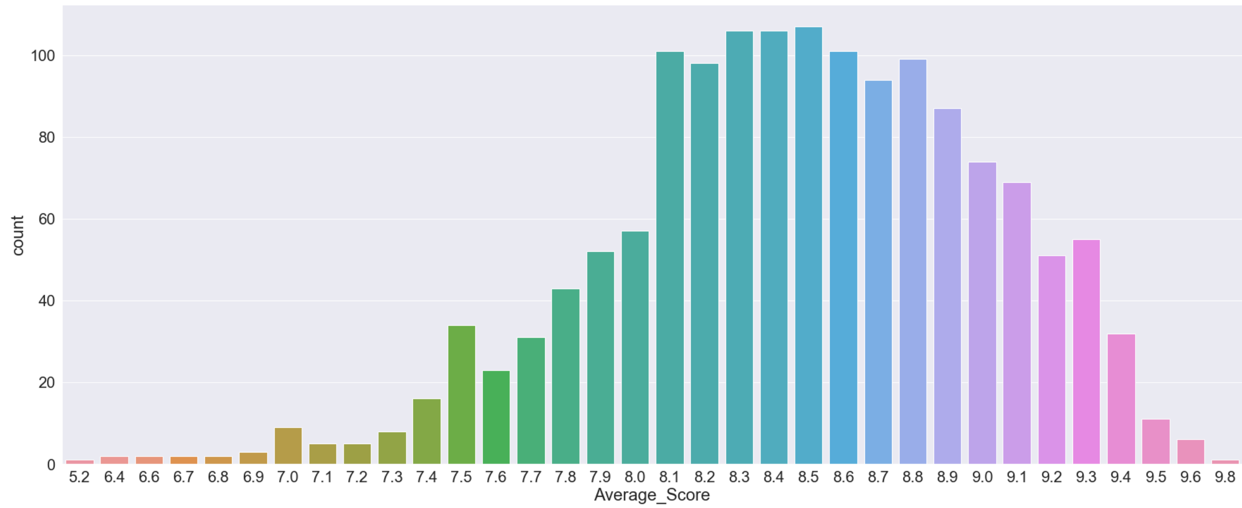
All 17 fields in the dataset are described below:

- Hotel_Address: Address of the hotel.
- Review_Date: Date when reviewer posted the corresponding review.
- Average_Score: Average Score of the hotel, calculated based on the latest comments in the last year.
- Hotel_Name: Name of the hotel.
- Reviewer_Nationality: Nationality of the reviewer.
- Negative_Review: Negative/bad things the reviewer wrote about the hotel. If the reviewer didn't give a negative review, its value shows 'No Negative'.
- Review_Total_Negative_Word_Counts: Total words in the negative review.
- Positive_Review: Positive/good things the reviewer wrote the hotel. If the reviewer didn't give a negative review, its value shows 'No Positive'.
- Review_Total_Positive_Word_Counts: Total words in the positive review.
- Reviewer_Score: A score the reviewer has given to the hotel, based on his/her experience.
- Total_Number_of_Reviews_Reviewer_Has_Given: Number of Reviews the reviewer has given in the past.
- Total_Number_of_Reviews: Total number of valid reviews the hotel has.
- Tags: Tags reviewer gave the hotel.
- days_since_review: Duration between the review date and scrape date.
- Additional_Number_of_Scoring: There are also some guests who just made a scoring on the service rather than a review. This number indicates how many valid scores without review in there.
- lat: Latitude of the hotel.
- lng: longtitude of the hotel.

We take closer look in Average_Score, Hotel_Name, Negative_Review, Review_Total_Negative_Word_Counts, Positive_Review, Review_Total_Positive_Word_Counts, Reviewer_Score for Sentiment Analysis. Since both negative reviews and positive reviews are already split and provided by the dataset, so starting with binary classification would be natural choice for this project.

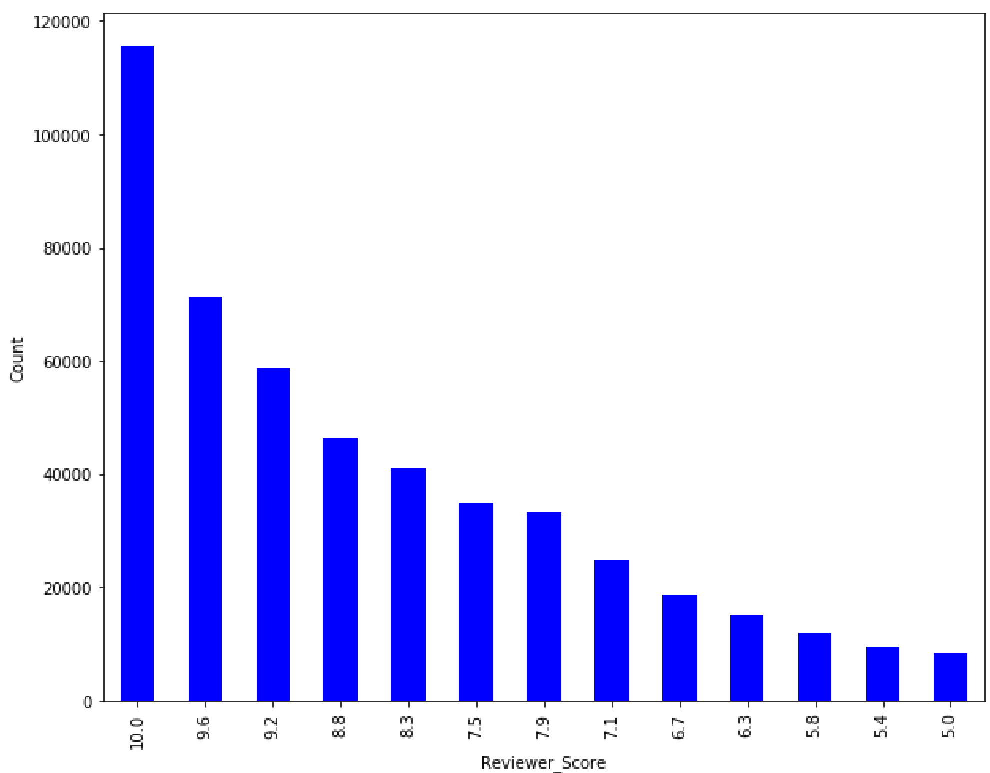
And geographical information including Hotel_Address, lat, lng might be helpful for data visualization in the later phase of the project.

Average Score Distribution:
The histogram below represents the distribution of Hotel Average_Score. It ranges from 5.2 to 9.8, and the most common average score are between 8.1 to 9.1. Since Average_Score is calculated based on all Reviewer_Scores from the previous year, it tends to "balance" some extreme values, so we see a relatively smooth and "normal" distribution.

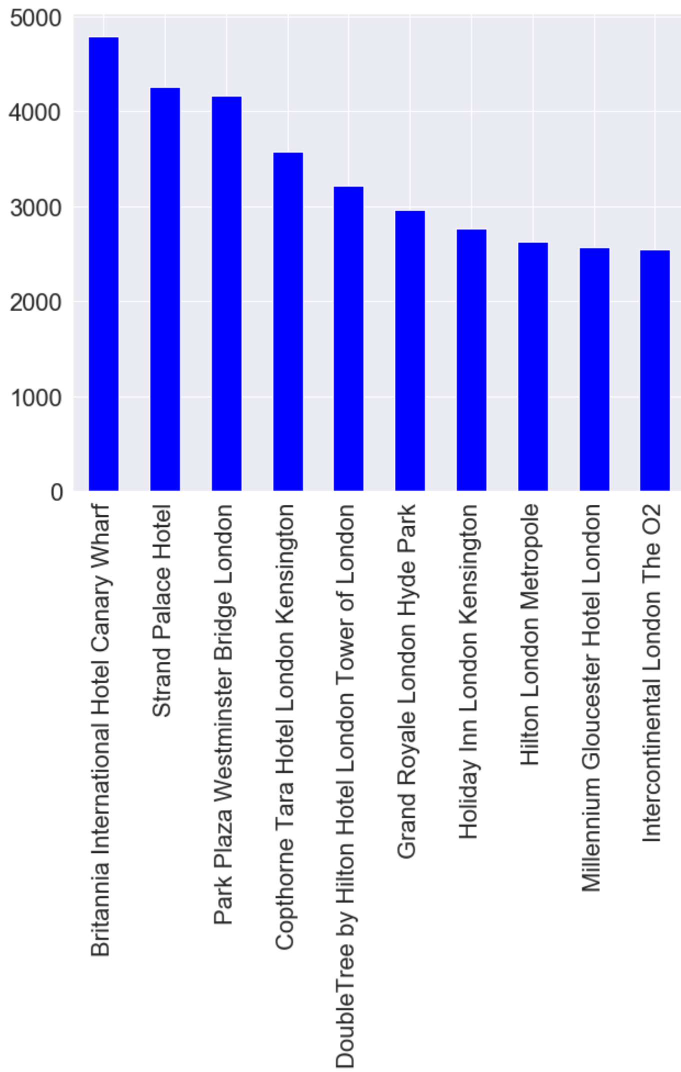



Reviewer Score Distribution:

The histogram below represents the top 5 distribution of Hotel Reviewer_Score. The range of review_score is from 1.0 to 10.0.




Top 10 Most Reviewed Hotels:




Word could distribution for Review_Nationality:


## Simple Binary Classification (data_process.ipynb)

1. Handle duplicated and missing values

We start with removing duplicated values and checking missing values. Most missing values are Hotel_Names,  latitudes and longitudes of hotels, we use information provided by http://latlong.org/ to manually fill in those blanks for future geographical display.

```
###Latitude information of Hotels
loc_lat = {'Fleming s Selection Hotel Wien City':48.209270,
         'Hotel City Central':48.2136,
         'Hotel Atlanta':48.210033,
         'Maison Albar Hotel Paris Op ra Diamond':48.875343,
         'Hotel Daniel Vienna':48.1888,
         'Hotel Pension Baron am Schottentor':48.216701,
         'Austria Trend Hotel Schloss Wilhelminenberg Wien':48.2195,
         'Derag Livinghotel Kaiser Franz Joseph Vienna':48.245998,
         'NH Collection Barcelona Podium':41.3916,
         'City Hotel Deutschmeister':48.22088,
         'Hotel Park Villa':48.233577,
         'Cordial Theaterhotel Wien':48.209488,
         'Holiday Inn Paris Montmartre':48.888920,
         'Roomz Vienna':48.186605,
         'Mercure Paris Gare Montparnasse':48.840012,
         'Renaissance Barcelona Hotel':41.392673,
         'Hotel Advance':41.383308}
```

```
###Longitude information of Hotels
loc_lng ={'Fleming s Selection Hotel Wien City':16.353479,
         'Hotel City Central':16.3799,
         'Hotel Atlanta':16.363449,
         'Maison Albar Hotel Paris Op ra Diamond':2.323358,
         'Hotel Daniel Vienna':16.3840,
         'Hotel Pension Baron am Schottentor':16.359819,
         'Austria Trend Hotel Schloss Wilhelminenberg Wien':16.2856,
         'Derag Livinghotel Kaiser Franz Joseph Vienna':16.341080,
         'NH Collection Barcelona Podium':2.1779,
         'City Hotel Deutschmeister':16.36663,
         'Hotel Park Villa':16.345682,
         'Cordial Theaterhotel Wien':16.351585,
         'Holiday Inn Paris Montmartre':2.333087,
         'Roomz Vienna':16.420643,
         'Mercure Paris Gare Montparnasse':2.323595,
         'Renaissance Barcelona Hotel':2.167494,
         'Hotel Advance':2.162828}
```

2. Stemming and Tokenization

We install and use **NLTK** libraries to do basic data pre-processing. When apply built-in stopwords in NLTK library, the word “us” shows in the most 20 common words in negative words. In order to remove a stop word such as “us”, apply a pre-downloaded “stopwords.txt” file. Then stem the word by applying the out-of-box PorterStemmer() from NLTK.

```python
###nltk
##removing stop words
import re
import time
import nltk
from collections import Counter
from nltk import word_tokenize
from nltk.corpus import stopwords ###not work well
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
start_time = time.time()
text = text_df['reviews'].values
print("Start removing stop words")
stop = set(stopwords.words('english'))  ##not work well
stop_words = [word.strip("\n") for word in open('stopwords.txt').readlines()]
#len(stop)
words = []
summary = []
all_pos_words = []
all_neg_words = []
for i in range(0,len(text)):
    if type(text[i]) == type('') :
        sentence = text[i]
        sentence = re.sub("[^a-zA-Z]"," ", sentence)
        buffer_sentence = [k for k in sentence.split() if k not in stop]
        buffer_sentence_n = [k for k in sentence.split() if k not in stop_words]
        word = ''
        for j in buffer_sentence:
            if len(j) >= 2:
                if i <= (len(text)/2):
                    all_pos_words.append(j)
                else:
                    all_neg_words.append(j)
                word +=' '+j
        summary.append(word)
print("performing stemming")
porter = PorterStemmer()
for i in range(0,len(summary)):
    summary[i] = porter.stem(summary[i])
print("--- %s seconds ---" % (time.time() - start_time))
```

```
Start removing stop words
performing stemming
--- 158.507291079 seconds ---
```

3. Label positive and negative reviews

We clean up the corpus by eliminating white space, removing numbers and coverting all words in reviews to lower case. For simplicity's sake, each positive/negative review is treated as a document to be classified. We assign each positive review numeric value 1 and each negative review numeric value 0. One thing worth noting is we treat “no positive” in positive review as a 100% negative review, “no negative” in negative review as a 100% positive review and treat “nothing” in positive or negative review as negative / positive review respectively. The other case is to treat “everything” in positive/negative review as negative/positive review respectively. Based on these labels, we have prepared 477,358 negative reviews and 327,667 positive reviews for the next step, classifier comparison and evaluation.

```
df['pos_count'] = 0
df['neg_count'] = 0
##remove space and make each word lowercases
df['Negative_Review'] = [x.lower().strip() for x in df['Negative_Review']]
df['Positive_Review'] = [x.lower().strip() for x in df['Positive_Review']]
#df['Negative_Review'].head()
```

```
##if 'nothing' 'no positive' in Positive_Review, turn positive review to negative
df['neg_count'] = df.apply(lambda x: 1 if x['Positive_Review'] == 'no positive' or
                             x['Positive_Review'] == 'nothing' or
                            x['Negative_Review'] == 'everything'
                            else x['pos_count'], axis = 1)
```

```
##if 'nothing' 'no negative' in Negative_Review, turn positive review to positive
df['pos_count'] = df.apply(lambda x: 1 if x['Negative_Review'] == 'no negative' or
                            x['Negative_Review'] == 'nothing' or
                            x['Positive_Review'] == 'everything'
                            else x['neg_count'], axis = 1)
```

```
df.pos_count.value_counts()
```

```
0    327667
1    187545
Name: pos_count, dtype: int64
```

```
df.neg_count.value_counts()
```

```
0    477358
1     37854
Name: neg_count, dtype: int64
```

```
pos_reviews = df['Positive_Review'].values
pos_reviews = pos_reviews.tolist()
neg_reviews = df['Negative_Review'].values
neg_reviews = neg_reviews.tolist()
#pos_reviews
#neg_reviews
total_text = pos_reviews + neg_reviews
#total_text
```

```
#pos_reviews
```

```
score = ['positive' for i in range(len(pos_reviews))]
score += ['negative' for i in range(len(neg_reviews))]
for i in range(len(score)):
    if score[i] == 'positive':
        score[i] = 1
    else:
        score[i] = 0
```

```
text_df = pd.DataFrame()
text_df['reviews'] = total_text
text_df['score'] = score
text_df.head()
```

| | reviews | score |
|---|---|---|
| 0 | only the park outside of the hotel was beautiful | 1 |
| 1 | no real complaints the hotel was great great l... | 1 |
| 2 | location was good and staff were ok it is cute... | 1 |
| 3 | great location in nice surroundings the bar an... | 1 |
| 4 | amazing location and building romantic setting | 1 |

We’re able to extract 20 most common positive words and 20 most common negative words and their frequencies:

```
print("Most common new positive words: ", freq_pos_n.most_common(20))
print("Most common new negative words: ", freq_neg_n.most_common(20))
```

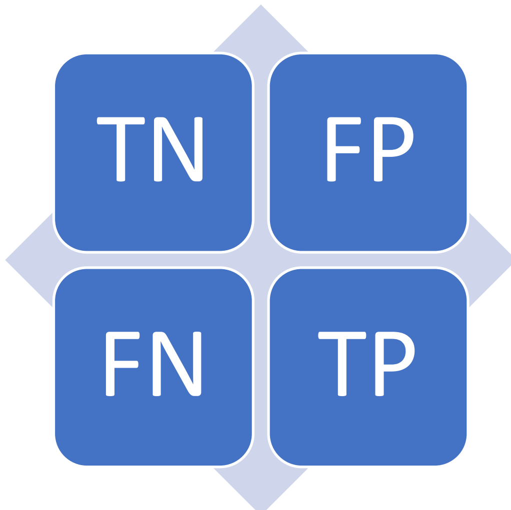
```
('Most common new positive words: ', [('staff', 194387), ('location', 192645), ('room', 140651), ('hotel', 125218), ('good', 11
2201), ('great', 105531), ('friendly', 85273), ('breakfast', 84524), ('helpful', 76102), ('nice', 69379), ('clean', 66859), ('e
xcellent', 62229), ('comfortable', 59903), ('bed', 49881), ('rooms', 40325), ('positive', 36445), ('lovely', 35073), ('stay', 3
2008), ('close', 30936), ('station', 29262)])
('Most common new negative words: ', [('room', 175835), ('negative', 129312), ('hotel', 74625), ('breakfast', 58410), ('small',
49837), ('staff', 39467), ('rooms', 34776), ('bed', 29819), ('bit', 27521), ('bathroom', 26568), ('didn', 26441), ('night', 240
71), ('shower', 21283), ('good', 20789), ('did', 20086), ('service', 19305), ('bar', 19115), ('time', 17458), ('stay', 17410),
('reception', 16625)])
```

Their word clouds are generated below:

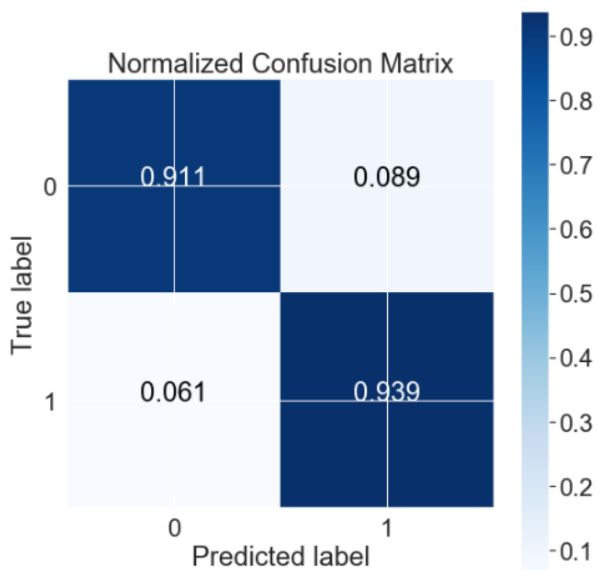


4. Classification Evaluation

We explore various classifiers from Scikit-Learn library and we look at their Precision, Recall, Accuracy score and run-time to ensure a both efficient and accurate classification. For each approach, we divide corpus into training and test dataset, select optimal parameters and create a confusion matrix illustrated as below. We also give a brief overview for each approach and please refer to our technical review for more details.




- Naïve Bayes
  A Naïve Bayes classifier is a family of probabilistic algorithms, which uses Bayes' theorem in the classifier's decision rule, with an independent assumption between features.



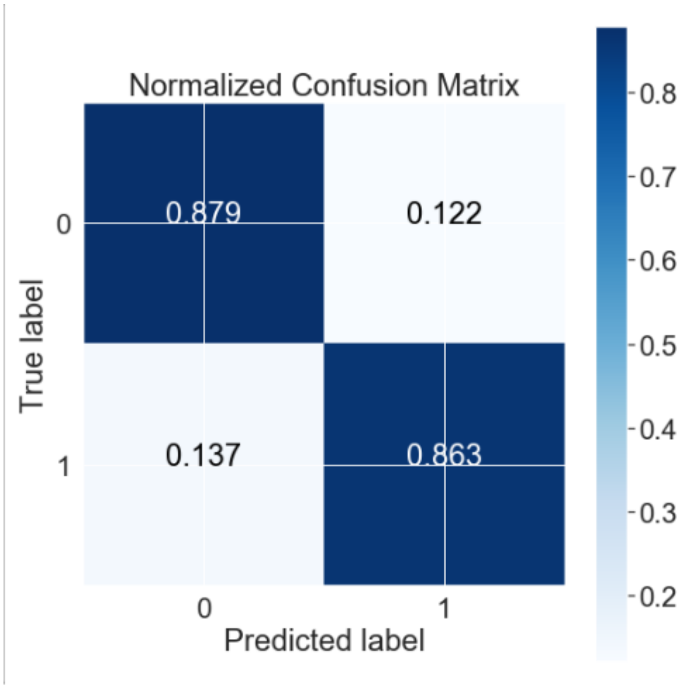

```
tn = conf_NB[0, 0]
fp = conf_NB[0, 1]
fn = conf_NB[1, 0]
tp = conf_NB[1, 1]
precision = 100 * float(tp) / (tp + fp)
recall = 100 * float(tp) / (tp + fn)
accuracy = 100 * float(tp + tn) / len(y_test)
#accuracy
#precision
#recall
```

```
print("Accuracy for Naive Bayes is {}%".format(round(accuracy, 2)))
print("Precision for Naive Bayes is {}%".format(round(precision, 2)))
print("Recall for Naive Bayes is {}%".format(round(recall, 2)))
```

```
Accuracy for Naive Bayes is 92.48%
Precision for Naive Bayes is 91.34%
Recall for Naive Bayes is 93.86%
```

- Logistic Regression

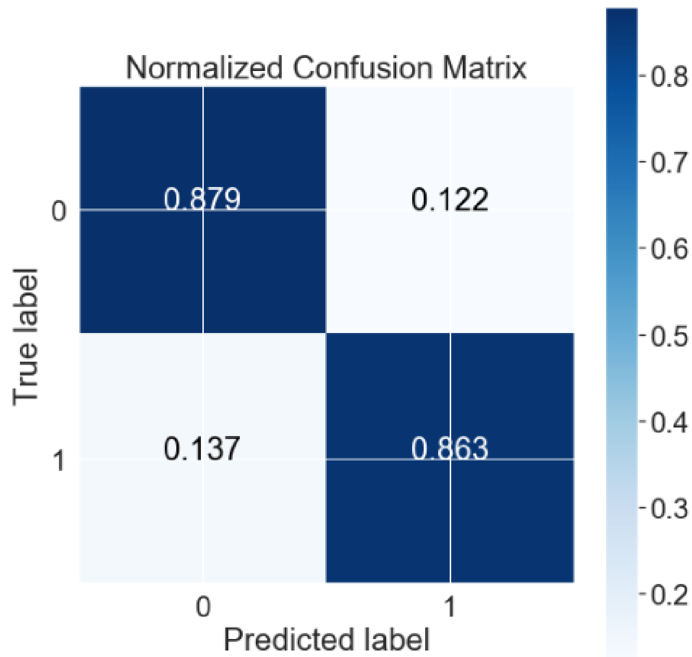
Logistic Regression is also common to solve Binary Classification problem. The goal of Binary Classification is thus to find a model that can best predict the probability of a discrete outcome (notated as 1 or 0, for the “positive” or “negative” classes), based on a set of explanatory input features related to that outcome.




```
tn_2 = conf_log_reg[0, 0]
fp_2 = conf_log_reg[0, 1]
fn_2 = conf_log_reg[1, 0]
tp_2 = conf_log_reg[1, 1]
precision_2 = 100 * float(tp_2) / (tp_2 + fp_2)
recall_2 = 100 * float(tp_2) / (tp_2 + fn_2)
accuracy_2 = 100 * float(tp_2 + tn_2) / len(y_test)
#accuracy_2
#precision_2
#recall_2
print("Accuracy for Logistic Regression is {}%".format(round(accuracy_2, 2)))
print("Precision for Logistic Regression is {}%".format(round(precision_2, 2)))
print("Recall for Logistic Regression is {}%".format(round(recall_2, 2)))
```

```
Accuracy for Logistic Regression is 87.08%
Precision for Logistic Regression is 87.65%
Recall for Logistic Regression is 86.3%
```

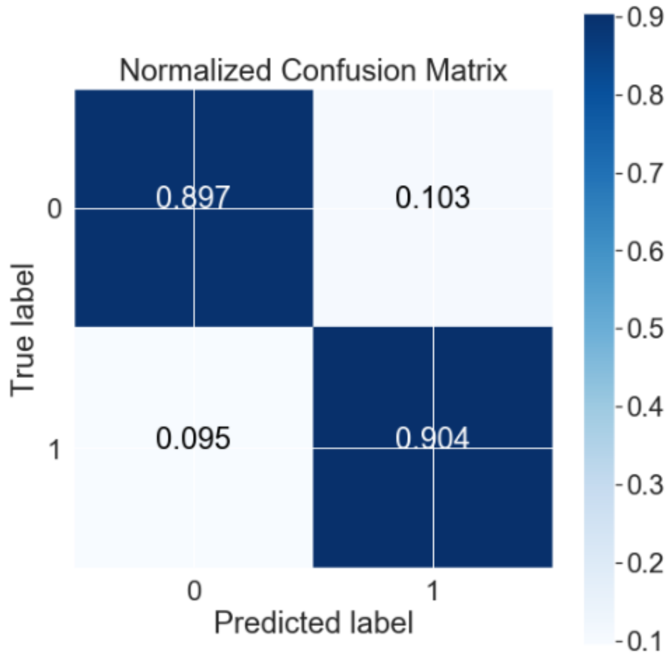
- Support Vector Machine (SVM)
  A SVM is a classifier which uses annotated data for training to construct an optimal separating hyperplane/line in a multi-dimensional space which can be used to categorize new samples data into different groups. It is one of the key machine learning methods widely used for Sentiment Analysis.




```
tn_3 = conf_SVM[0, 0]
fp_3 = conf_SVM[0, 1]
fn_3 = conf_SVM[1, 0]
tp_3 = conf_SVM[1, 1]
precision_3 = 100 * float(tp_3) / (tp_3 + fp_3)
recall_3 = 100 * float(tp_3) / (tp_3 + fn_3)
accuracy_3 = 100 * float(tp_3 + tn_3) / len(y_test)
#accuracy_3
#precision_3
#recall_3
print("Accuracy for SVM is {}%".format(round(accuracy_3, 2)))
print("Precision for SVM is {}%".format(round(precision_3, 2)))
print("Recall for SVM is {}%".format(round(recall_3, 2)))
```

```
Accuracy for SVM is 87.69%
Precision for SVM is 89.59%
Recall for SVM is 85.27%
```

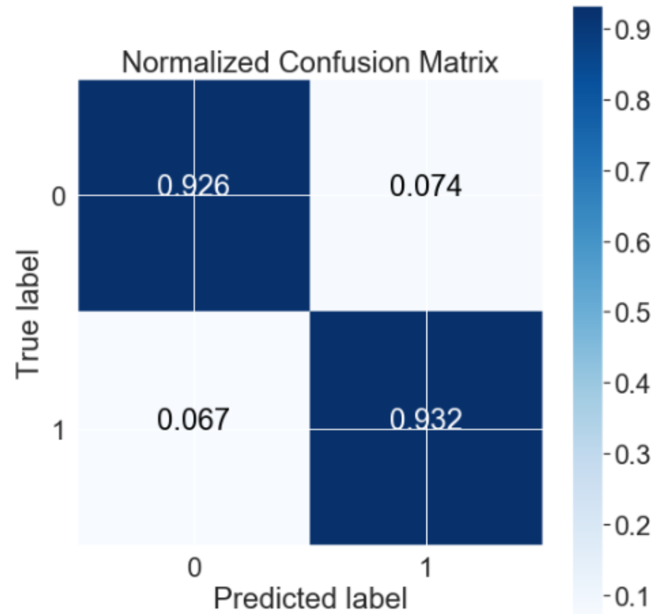
- Decision trees
  Decision tree is a non-parametric learning method that predicts the value of a target variable by learning decision rules. Tree models where the target variable can take a discreate set of values are called classification trees and leaves represent class labels and breaches represent connections of features that lead to those labels. One of the advantages of decision trees is to learn inherent rules available in the dataset that are not available to the user.




```
tn_4 = conf_dec_tree[0, 0]
fp_4 = conf_dec_tree[0, 1]
fn_4 = conf_dec_tree[1, 0]
tp_4 = conf_dec_tree[1, 1]
precision_4 = 100 * float(tp_4) / (tp_4 + fp_4)
recall_4 = 100 * float(tp_4) / (tp_4 + fn_4)
accuracy_4 = 100 * float(tp_4 + tn_4) / len(y_test)
#accuracy_4
#precision_4
#recall_4
print("Accuracy for Decision Tree is {}%".format(round(accuracy_4, 2)))
print("Precision for Decision Tree is {}%".format(round(precision_4, 2)))
print("Recall for Decision Tree is {}%".format(round(recall_4, 2)))
```

```
Accuracy for Decision Tree is 90.07%
Precision for Decision Tree is 89.77%
Recall for Decision Tree is 90.44%
```

Naïve Bayes classifier has proved again that the simplest solutions are usually the most powerful ones. Despite other more advanced techniques, Naïve Bayes has achieved the highest accuracy score and reasonable run-time. Our next decision is how to continually improve Naïve Bayes classifier. In order to do so, we test bi-gram classifier and experimental results suggest that bigrams can substantially raise the performance of classifier especially for Recall.




```
tn_5 = conf_NB_Bi[0, 0]
fp_5 = conf_NB_Bi[0, 1]
fn_5 = conf_NB_Bi[1, 0]
tp_5 = conf_NB_Bi[1, 1]
precision_5 = 100 * float(tp_5) / (tp_5 + fp_5)
recall_5 = 100 * float(tp_5) / (tp_5 + fn_5)
accuracy_5 = 100 * float(tp_5 + tn_5) / len(y_test)
#accuracy_5
#precision_5
#recall_5
print("Accuracy for Bi-gram Naive Bayes is {}%".format(round(accuracy_5, 2)))
print("Precision for Bi-gram Naive Bayes is {}%".format(round(precision_5, 2)))
print("Recall for Bi-gram Naive Bayes is {}%".format(round(recall_5, 2)))
```

```
Accuracy for Bi-gram Naive Bayes is 92.92%
Precision for Bi-gram Naive Bayes is 92.63%
Recall for Bi-gram Naive Bayes is 93.24%
```

## Sentiment Analysis (aspect_analysis_data_process.ipynb & aspect_analysis.ipynb)

We switch to NLTK for further Sentiment Analysis, because 1) NLTK comes with all ideal built-in functions we need for Sentiment Analysis (textual tokenization, parsing, classification, stemming, tagging, semantic reasoning etc.) and 2) all of NLTK classifiers work with "featstructs", which is a simple dictionary mapping a feature name to a feature value. It allows us to use simplified bag of words model where every existing word is a feature name with a value of True. From previous step, we have known that Naïve Bayes performs best with our dataset, so we first create our own classification model using Naïve Bayes classifier again. 70% reviews are used as training data for the classifier and the remaining 30% reviews are used to test and calculate accuracy score. Contrast to the previous classification where we use all the words as features, here we only use 10,000 most informative words for positive reviews and negative reviews. Save each word and its corresponding sentiment score to "pos_word_sen_score.csv" and "neg_word_sen_score.csv" for future use. Now we're ready to put this preparation into use for three applications.

```python
probdist = classifier_NB._feature_probdist
word_list = []
word_score = []
sentiment = []
for (name, value) in classifier_NB.most_informative_features(10000):
    def label_prob(l):
        return probdist[l,name].prob(value)
    labels = sorted([l for l in classifier_NB._labels if value in probdist[l, name].samples()],
                    key = label_prob)
    if len(labels) == 1:
        continue
    l0 = labels[0]
    l1 = labels[-1]
    if probdist[l0, name].prob(value) == 0:
        ration = "INF"
    else:
        ration = '%8.1f' % (probdist[l1, name].prob(value) / probdist[l0, name].prob(value))
    sentiment.append(int(l1))
    word_list.append(name)
    word_score.append(float(ration))
#len(word_list)
```

```python
word_sentiment_score = pd.DataFrame({'word': word_list, 'sentiment': sentiment, 'score': word_score})
word_sentiment_score.head()
neg_word_sen_score = word_sentiment_score[word_sentiment_score.sentiment == 0]
#display(word_sentiment_score[word_sentiment_score['sentiment'] == 1])
neg_word_sen_score.head()
neg_word_sen_score.shape
```

```
(6441, 3)
```

```python
pos_word_sen_score = word_sentiment_score[word_sentiment_score.sentiment == 1]
pos_word_sen_score.head()
pos_word_sen_score.shape
```

```
(3559, 3)
```

```python
neg_word_sen_score.to_csv('neg_word_sen_score.csv', index=False)
pos_word_sen_score.to_csv('pos_word_sen_score.csv', index=False)
```

In order to compare each hotel based on various aspects, our project calculates each positive sentiment score and negative sentiment score based on above method for each positive and negative review (see details in “data_process.ipynb”). Then save each score to “pos_score.txt” and “neg_score.txt”. This step may take longer time.

```python
pos_score = []
start_time = time.time()
for i in range(len(pos_reviews)):
    pos_score.append(pos_sentiment_score(pos_reviews[i]))
print('pos_score %s seconds' % (time.time() - start_time))
pos_score[:5]
```

```
pos_score 3108.43393493 seconds

[23.1, 135.7, 57.300000000000004, 63.599999999999994, 43.6]
```

```python
###save to file
np.savetxt('pos_score.txt', pos_score, delimiter=',')
```

```python
neg_score = []
start_time = time.time()
for i in range(len(neg_reviews)):
    neg_score.append(neg_sentiment_score(neg_reviews[i]))
print('neg_score %s seconds' % (time.time() - start_time))
neg_score[:5]
###save to file
np.savetxt('neg_score.txt', neg_score, delimiter=',')
```

```
neg_score 3423.66275191 seconds
```

```python
data = pd.read_pickle('Filling_nans')
##print(len(data), len(pos_reviews), len(neg_reviews))  (515212, 515212, 515212)
new_dataset = pd.DataFrame({'Hotel_name': data['Hotel_Name'], 'Avg_score': data['Average_Score'], 'pos': pos_score,
                           'neg':  neg_score})
new_dataset.head()
```

| | Avg_score | Hotel_name | neg | pos |
|---|---|---|---|---|
| **0** | 7.7 | Hotel Arena | 465.1 | 23.1 |
| **1** | 7.7 | Hotel Arena | 0.0 | 135.7 |
| **2** | 7.7 | Hotel Arena | 63.2 | 57.3 |
| **3** | 7.7 | Hotel Arena | 466.0 | 63.6 |
| **4** | 7.7 | Hotel Arena | 141.8 | 43.6 |

## Application 1: Strength of Sentiment in Reviews

This application will estimate the strength of positive and negative sentiment in reviews. It takes a review as input and return a positive sentiment strength (ranging from 0) and a negative sentiment strength (ranging from 0) by calling pos_sentiment_score() and neg_sentiment_score() respectively. In this way, hotels will better understand customers' experience by extracting their emotional tone from the reviews they post and use this insight to improve their business and gain competitive advantages. See details in "data_process.ipynb".

```
toktok = ToktokTokenizer()
stop_words_nltk = set(stopwords.words('english'))
def word_process(text):
    clean_words = re.sub("[^a-zA-Z]"," ", text)
    clean_words = clean_words.lower()
    words = word_tokenize(clean_words)
    words = [toktok.tokenize(k) for k in sent_tokenize(clean_words)]
    result = []
    if not words:
        pass
    else:
        for w in words[0]:
            if w not in stop_words_nltk:
                result.append(w)
    return result
```

```
set_of_pos_word = set(pos_word_sen_score.word) - set(['no', 'negative', 'positive'])
set_of_neg_word = set(neg_word_sen_score.word) - set(['no', 'negative', 'positive'])
def pos_sentiment_score(review):
    pos = 0
    clean = word_process(review)
    for w in clean:
        if w in set_of_pos_word:
            pos += pos_word_sen_score[pos_word_sen_score['word'] == w].score.iloc[0]
    return pos
def neg_sentiment_score(review):
    neg = 0
    clean = word_process(review)
    for w in clean:
        if w in set_of_neg_word:
            neg += neg_word_sen_score[neg_word_sen_score['word'] == w].score.iloc[0]
    return neg
```

Test Case (test_case.ipynb):

Input:

My room was dirty and I was afraid to walk barefoot on the floor which looked as if it was not cleaned in weeks White furniture which looked nice in pictures was dirty too and the door looked like it was attacked by an angry dog My shower drain was clogged and the staff did not respond to my request to clean it On a day with heavy rainfall a pretty common occurrence in Amsterdam the roof in my room was leaking luckily not on the bed you could also see signs of earlier water damage I also saw insects running on the floor Overall the second floor of the property looked dirty and badly kept On top of all of this a repairman who came to fix something in a room next door at midnight was very noisy as were many of the guests I understand the challenges of running a hotel in an old building but this negligence is inconsistent with prices demanded by the hotel On the last night after I complained about water damage the night shift manager offered to move me to a different room but that offer came pretty late around midnight when I was already in bed and ready to sleep.

```
test_text1 = 'My room was dirty and I was afraid to walk barefoot on the floor which looked as if it was not cleaned in weeks Whi
```

Output:

```
print("The positive sentiment score of the given text is %.2f" %pos_sentiment_score(test_text1),
      "The negative sentiment score of the given text is %.2f" %neg_sentiment_score(test_text1))
```

```
('The positive sentiment score of the given text is 31.50', 'The negative sentiment score of the given text is 466.00')
```

The given test review is classified as a **negative review** since the neg_sentiment_score is much larger than pos_sentiment_score. And since this review is coped from column "Negative_Review", the result matches the right categorization.

Input:

Great location in nice surroundings the bar and restaurant are nice and have a lovely outdoor area The building also has quite some character

```
test_text2 = 'Great location in nice surroundings the bar and restaurant are nice and have a lovely outdoor area The building als
```

Output:

```
print("The positive sentiment score of the given text is %.2f" %pos_sentiment_score(test_text2),
      "The negative sentiment score of the given text is %.2f" %neg_sentiment_score(test_text2))
```

```
('The positive sentiment score of the given text is 63.60', 'The negative sentiment score of the given text is 0.00')
```

The given test review text is classified as a **100% positive review** since the neg_sentiment_score is 0. And since this review is copied form column "Positive_Review", the result matches its right categorization.

The two returned scores tell us how negative or positive a given review is, in other words, how dissatisfied or satisfied a customer is. The two test cases show that our tool works as desired.

## Application 2: Reviewer Score Prediction

As an extension of application 1, application 2 takes two reviews (no need to specify which one is positive and which one is negative, but they need to be different) as input and predicts Reviewer_Score using the correlation between Reviewer_Score and strengths of positive and negative sentiment (from application 1) in reviews by calling Reviewer_score_cal(). This application can be beneficial to hotels in some scenarios, especially where customers forget leaving an overall score. With our tool at their disposal, it’s easy and quick for hotel managers to generate a numeric feedback out of mixed reviews.

We use built-in linear regression model from Scikit-Learn library and Mean Absolute Error, Mean Squared Error, Root Mean Squared Error to evaluate its performance.  See model details in “reviewer_score.ipynb”.

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,random_state = 42)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

```python
print(regressor.coef_, regressor.intercept_)
```

```
(array([ 0.01183668, -0.01697772]), 8.24393084976765)
```

```python
y_pred = regressor.predict(X_test)
comparison = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
comparison[:5]
```

| | Actual | Predicted |
|---|---|---|
| **499860** | 8.3 | 8.653194 |
| **453739** | 7.9 | 8.443674 |
| **484096** | 9.6 | 8.172326 |
| **415438** | 10.0 | 9.280633 |
| **248620** | 4.6 | 6.317105 |

```python
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
('Mean Absolute Error:', 1.0824430198662658)
('Mean Squared Error:', 1.9188871956284612)
('Root Mean Squared Error:', 1.3852390391656095)
```

```python
review_score_model = [regressor.coef_[0], regressor.coef_[1], regressor.intercept_]
review_score_model
```

```
[0.011836679330565333, -0.016977716121541978, 8.24393084976765]
```

```python
model_para = pd.DataFrame({'para': review_score_model})
model_para.to_pickle('reviewer_score_model_para')
```

```python
model_para
```

| | para |
|---|---|
| **0** | 0.011837 |
| **1** | -0.016978 |
| **2** | 8.243931 |

```python
model_para = pd.read_pickle('reviewer_score_model_para') ###read model parameters
model_para = model_para['para'].values.tolist()
model_para
```

```
[0.011836679330565333, -0.016977716121541978, 8.24393084976765]
```

```python
def Reviewer_score_cal(a, b):
    if a == b:
        return('Error: Given two texts are the same. Please give two different texts')
    pos_score_a = pos_sentiment_score(a)
    neg_score_a = neg_sentiment_score(a)
    pos_score_b = pos_sentiment_score(b)
    neg_score_b = neg_sentiment_score(b)
    if pos_score_a > neg_score_a and neg_score_b > pos_score_b:
        pos = pos_score_a
        neg = neg_score_b
        result = model_para[0] * pos + model_para[1] * neg + model_para[2]
        return result
    elif neg_score_a > pos_score_a and pos_score_b > neg_score_b:
        pos = pos_score_b
        neg = neg_score_a
        result = model_para[0] * pos + model_para[1] * neg + model_para[2]
        return result
    elif pos_score_a > neg_score_a and pos_score_a > neg_score_a:
        return('Error: the given two reviews are both positive reviews')
    elif pos_score_a < neg_score_a and pos_score_a < neg_score_a:
        return('Error: the given two reviews are both negative reviews')
```

Test Case (test_case.ipynb):

Input:

We continue to use two reviews from Application 1.

Output:

```python
Reviewer_score_cal(test_text1, test_text2)
```

```
1.0851279425530453
```

```python
print('The predicted Reviewer Score of the two given texts is %.2f' %Reviewer_score_cal(test_text1, test_text2))
```

```
The predicted Reviewer Score of the two given texts is 1.09
```

If these two reviews are given by the same reviewer to a certain hotel, we could predict the Reviewer Score for this hotel is 1.09 on the scale of 1 to 10 based on this experience.

## Application 3: Hotel Ranking List Based on Aspects

There are three different levels of Sentiment Analysis: Document-based, Sentence-based and Aspect-based. Aspect-based Sentiment Analysis has gained increasing popularity in both industry and academia, but only recently it has entered the domain of tourism. Please refer to our technology review for a detailed review of existing research and trendy applications.

In our work, we implement an Aspect-based opinion miner for hotel reviews and find important aspects for positive and negative reviews, based on which we rank all hotels by Average_Score and Aspect-based sentiment analysis on reviews. First, we collect top 20 common words in positive reviews and negative reviews and save them in two .txt files called “freq_pos_20.txt” and “freq_neg_20.txt” respectively. By deeply looking at the 20 words from each file, the top 5 aspects are "staff", "location", "room", "breakfast", "bed" for positive and "room", "breakfast", "staff", "bed", "bathroom" for negative.

First apply the previous calculated positive sentiment score and negative sentiment score and group all the sentiment scores and Average_Score by “Hotel_Name”. Then save each aspect related data to pickle file.

```
pos_score = []
neg_score = []
pos_file = open('pos_score.txt', 'r')
neg_file = open('neg_score.txt', 'r')
for line in pos_file.readlines():
    pos_score.append(float(line))
for line in neg_file.readlines():
    neg_score.append(float(line))
len(pos_score), len(neg_score)
```

```
(515212, 515212)
```

```
import pandas as pd
data = pd.read_pickle('Filling_nans')
##print(len(data), len(pos_reviews), len(neg_reviews))  (515212, 515212, 515212)
new_dataset = pd.DataFrame({'Hotel_name': data['Hotel_Name'], 'Avg_score': data['Average_Score'], 'pos': pos_score,
                            'neg':  neg_score, 'Pos_Reviews': data['Positive_Review'], 'Neg_Reviews': data['Negative_Rev
new_dataset['Neg_Reviews'] = [x.lower().strip() for x in new_dataset['Neg_Reviews']]
new_dataset['Pos_Reviews'] = [x.lower().strip() for x in new_dataset['Pos_Reviews']]
new_dataset.head()
```

| | Avg_score | Hotel_name | Neg_Reviews | Pos_Reviews | neg | pos |
|---|---|---|---|---|---|---|
| **0** | 7.7 | Hotel Arena | i am so angry that i made this post available ... | only the park outside of the hotel was beautiful | 465.1 | 23.1 |
| **1** | 7.7 | Hotel Arena | no negative | no real complaints the hotel was great great l... | 0.0 | 135.7 |
| **2** | 7.7 | Hotel Arena | rooms are nice but for elderly a bit difficult... | location was good and staff were ok it is cute... | 63.2 | 57.3 |
| **3** | 7.7 | Hotel Arena | my room was dirty and i was afraid to walk bar... | great location in nice surroundings the bar an... | 466.0 | 63.6 |
| **4** | 7.7 | Hotel Arena | you when i booked with your company on line yo... | amazing location and building romantic setting | 141.8 | 43.6 |

```
staff_pos_data = new_dataset[new_dataset['Pos_Reviews'].str.contains("staff")]
staff_pos_data.head()
staff_pos_data.to_pickle("staff_pos_data")
```

**staff** in negative reviews

```
staff_neg_data = new_dataset[new_dataset['Neg_Reviews'].str.contains("staff")]
staff_neg_data.head()
staff_neg_data.to_pickle("staff_neg_data")
```

**location** in positive reviews

```
loc_pos_data = new_dataset[new_dataset['Pos_Reviews'].str.contains("location")]
loc_pos_data.to_pickle("loc_pos_data")
```

If a customer has no interest in any optional aspects, the customer will get a ranking list only based on descending value of “Average_Score” by reading “overall_list.csv”.

List of hotel based on value of **Average_Score**

```
overall_data = pd.read_csv('hotel_score.csv')
overall_list = overall_data.sort_values('Average_Score', ascending = False)
overall_list = overall_list.drop(columns = ['index'])
overall_list.head()
```

| | Hotel_Name | Average_Score | count |
|---|---|---|---|
| **1203** | Ritz Paris | 9.8 | 28 |
| **481** | H10 Casa Mimosa 4 Sup | 9.6 | 116 |
| **472** | H tel de La Tamise Esprit de France | 9.6 | 61 |
| **3** | 41 | 9.6 | 103 |
| **772** | Hotel The Serras | 9.6 | 213 |

```
overall_list = overall_list.reset_index(drop = True)
overall_list.head()
```

```
overall_list.to_csv('overall_list.csv')
```

In order to compare hotel under each aspect, firstly read each aspect-related data from pick file then filter reviews based on aspect. Secondly, group positive sentiment score and negative sentiment score and Average_Score by each Hotel_Name and get mean values respectively. Ranking rules are as followings: if any two hotel has different Average_Score, rank higher Average_Score first; if two hotels have the same Average_Score, compare mean ratio by the following formula, rank higher mean ratio first.

$$mean\ ratio = \frac{mean(positive\ sentiment\ score) - mean(negative\ sentiment\ score)}{mean(positive\ sentiment\ score)}$$

```
staff_pos_data = pd.read_pickle('staff_pos_data')
staff_neg_data = pd.read_pickle('staff_neg_data')
loc_pos_data = pd.read_pickle('loc_pos_data') ###loc == location
rm_pos_data = pd.read_pickle('rm_pos_data') ##rm == room
rm_neg_data = pd.read_pickle('rm_neg_data')
bk_pos_data = pd.read_pickle('bk_pos_data') ##bk == breakfast
bk_neg_data = pd.read_pickle('bk_neg_data')
bed_pos_data = pd.read_pickle('bed_pos_data')
bed_neg_data = pd.read_pickle('bed_neg_data')
bath_neg_data = pd.read_pickle('bath_neg_data') ##bath == bathroom
```

## "staff" in positive reviews

Return a list of hotel descendingly by score of Avg_score and (mean(pos)-mean(neg))/(mean(pos)

```
staff_pos_list = staff_pos_data.groupby('Hotel_name').agg({'pos': 'mean', 'neg': 'mean', 'Avg_score': 'mean'})
staff_pos_list.head()
```

| | neg | pos | Avg_score |
|---|---|---|---|
| **Hotel_name** | | | |
| **11 Cadogan Gardens** | 20.365517 | 69.240230 | 8.7 |
| **1K Hotel** | 27.890244 | 68.829268 | 7.7 |
| **25hours Hotel beim MuseumsQuartier** | 16.299588 | 78.984362 | 8.8 |
| **41** | 11.659375 | 67.742188 | 9.6 |
| **45 Park Lane Dorchester Collection** | 12.014286 | 54.600000 | 9.4 |

```
staff_pos_list['score'] = (staff_pos_list.pos - staff_pos_list.neg) / staff_pos_list.pos
```

```
staff_pos_list.head()
```

| Hotel_name | neg | pos | Avg_score | score |
|---|---|---|---|---|
| **11 Cadogan Gardens** | 20.365517 | 69.240230 | 8.7 | 0.705872 |
| **1K Hotel** | 27.890244 | 68.829268 | 7.7 | 0.594791 |
| **25hours Hotel beim MuseumsQuartier** | 16.299588 | 78.984362 | 8.8 | 0.793635 |
| **41** | 11.659375 | 67.742188 | 9.6 | 0.827886 |
| **45 Park Lane Dorchester Collection** | 12.014286 | 54.600000 | 9.4 | 0.779958 |

```
staff_pos = staff_pos_list.sort_values(['Avg_score', 'score'], ascending = [False, False])
staff_pos.head()
```

| Hotel_name | neg | pos | Avg_score | score |
|---|---|---|---|---|
| **Ritz Paris** | 15.041667 | 60.441667 | 9.8 | 0.751137 |
| **H10 Casa Mimosa 4 Sup** | 13.511475 | 107.865574 | 9.6 | 0.874738 |
| **41** | 11.659375 | 67.742188 | 9.6 | 0.827886 |
| **Haymarket Hotel** | 11.886000 | 86.226000 | 9.6 | 0.862153 |
| **H tel de La Tamise Esprit de France** | 7.146341 | 91.500000 | 9.6 | 0.921898 |

Each ranking list is saved to .csv file for user to read directly. 'loc_pos_list.csv' is a ranking list of hotels has "location" in positive reviews. 'staff_pos_list.csv' is a ranking list of hotels has "staff" in positive reviews. 'rm_pos_list.csv' is a ranking list of hotels has "room" in positive reviews. 'bk_pos_list.csv' is a ranking list of hotels has "location" in positive reviews. 'bed_pos_list.csv' is a ranking list of hotels has "bed" in positive reviews. 'rm_neg_list.csv' is a ranking list of hotels has "room" in negative reviews. 'bk_neg_list.csv' is a ranking list of hotels has "breakfast" in negative reviews. 'staff_neg_list.csv' is a ranking list of hotels has "staff" in negative reviews. 'bed_neg_list.csv' is a ranking list of hotels has "bed" in negative reviews. 'bath_neg_list.csv' is a ranking list of hotels has "bathroom" in negative reviews. Then visualize the result by google map via "folium" library.

```
staff_pos= staff_pos_list.reset_index()
staff_pos.head()
```

| | Hotel_name | neg | pos | Avg_score | score |
|---|---|---|---|---|---|
| **0** | 11 Cadogan Gardens | 20.365517 | 69.240230 | 8.7 | 0.705872 |
| **1** | 1K Hotel | 27.890244 | 68.829268 | 7.7 | 0.594791 |
| **2** | 25hours Hotel beim MuseumsQuartier | 16.299588 | 78.984362 | 8.8 | 0.793635 |
| **3** | 41 | 11.659375 | 67.742188 | 9.6 | 0.827886 |
| **4** | 45 Park Lane Dorchester Collection | 12.014286 | 54.600000 | 9.4 | 0.779958 |

```
staff_pos.to_csv('staff_pos_list.csv')
```

```
loc_pos = loc_pos_list.reset_index()
loc_pos.to_csv('loc_pos_list.csv')
rm_pos = rm_pos_list.reset_index()
rm_pos.to_csv('rm_pos_list.csv')
bk_pos = bk_pos_list.reset_index()
bk_pos.to_csv('bk_pos_list.csv')
bed_pos = bed_pos_list.reset_index()
bed_pos.to_csv('bed_pos_list.csv')
rm_neg = rm_neg_list.reset_index()
rm_neg.to_csv('rm_neg_list.csv')
bk_neg = bk_neg_list.reset_index()
bk_neg.to_csv('bk_neg_list.csv')
staff_neg = staff_neg_list.reset_index()
staff_pos.to_csv('staff_neg_list.csv')
bed_neg = bed_neg_list.reset_index()
bed_neg.to_csv('bed_neg_list.csv')
bath_neg = bath_neg_list.reset_index()
bath_neg.to_csv('bath_neg_list.csv')
```

This way, hotel can address customers' different needs which is critical to their experience satisfaction. For a potential guest who has high standards on location, this tool will return top 20 hotels with highest average score and best location-related reviews. Hotel Ranking List by aspect-based sentiment score is extremely helpful when it comes to hotel decision-making process.  One question all the time is which hotel I should choose when there is a tie in their ratings. Our tool solves this problem by asking what matters most and rescues potential customers from a struggle to pick out useful comments from enormous amount of reviews.

Test Case (test_case.ipynb):

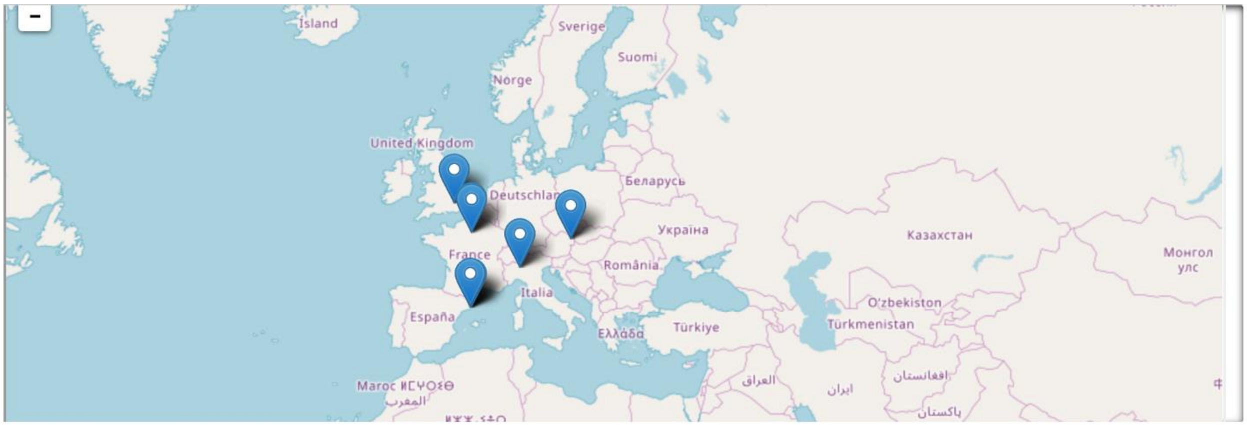
Assume a customer is interested in "location" from positive reviews.

```
import pandas as pd
loc_pos = pd.read_csv('loc_pos_list.csv')
loc_pos['Hotel_name'][:20]
```

```
0                                11 Cadogan Gardens
1                                         1K Hotel
2                     25hours Hotel beim MuseumsQuartier
3                                               41
4                     45 Park Lane Dorchester Collection
5                                       88 Studios
6                                9Hotel Republique
7                                A La Villa Madame
8              ABaC Restaurant Hotel Barcelona GL Monumento
9          AC Hotel Barcelona Forum a Marriott Lifestyle ...
10         AC Hotel Diagonal L Illa a Marriott Lifestyle ...
11                  AC Hotel Irla a Marriott Lifestyle Hotel
12                AC Hotel Milano a Marriott Lifestyle Hotel
13                  AC Hotel Paris Porte Maillot by Marriott
14                 AC Hotel Sants a Marriott Lifestyle Hotel
15         AC Hotel Victoria Suites a Marriott Lifestyle ...
16                             ADI Doria Grand Hotel
17                         ADI Hotel Poliziano Fiera
18                     ARCOTEL Kaiserwasser Superior
19                                ARCOTEL Wimberger
Name: Hotel_name, dtype: object
```

```
import folium
df = pd.read_pickle('Filling_nans')
loc_pos_hotel = loc_pos['Hotel_name'][:20]
loc_pos_data = df.loc[df['Hotel_Name'].isin(loc_pos_hotel)][["Hotel_Name","Hotel_Address",
                                                              'lat','lng']].drop_duplicates()
loc_pos_map = folium.Map(location = [52, 17], zoom_start = 1)
loc_pos_data.apply(lambda row:folium.Marker(location=[row["lat"], row["lng"]])
                                             .add_to(loc_pos_map), axis=1)

loc_pos_map
```




## Discussion

Tremendous amount of Hotel reviews are available on booking.com but it takes lots of time to read and analyze them, therefore for that purpose Sentiment Analysis is needed. The main contribution of our work can be summarized as below:

1. Based on relatively large amount of data

The corpus of 515k reviews is used to build and test review classifiers for Sentiment Analysis. It might take hours to run the whole script without cache.

2. Different classifiers and libraries are deployed and compared

We use different pre-processing strategies and machine learning approaches to determine the polarity of hotel reviews. In conclusion we determine that Bigram feature with Naïve Bayes classifier works best with our dataset.

3. Document-based Sentiment Analysis and Aspect-based Sentiment Analysis

By treating each positive/negative review as a document to be classified, our tool is able to associate customer feedback with overall sentiment scores telling hotels how happy or dissatisfied a customer is in general. To reach Sentiment Analysis' full potential, our tool takes different aspects of hotel experience into consideration and provides how people's sentiment varies when they are talking about room, staff, breakfast or location. There're some good opportunities for hotels to re-evaluate areas of opportunity and growth and better understand their clients.

4. Useful for both hotel managers and potential customers

There is always room for improvement as hotels strive to give customer to best possible experience. Determining important features/aspects expressed in online hotel reviews is vital for hotel managers as mentioned in 3. In addition, potential guests using this tool are equipped with a more powerful weapon when booking their hotels. They are provided a chance to filter same-rated hotels by a feature that matters most and a ranking list based on the feature.

## Future Work

In the end, how might this tool be improved? The answer lies in the ability to derive additional insights from the data. Some directions of future work include:

1. Aspect-based Sentiment Analysis and Reviewer_Score

Current work is being done to tie Reviewer_Score with overall sentiment score using linear regression. Furthermore, prediction on Reviewer_Score can be achieved though Aspect-based Sentiment Analysis and more advanced machine learning approaches. This will help answer questions like:

- How positive or negative opinions for each aspect contribute to overall sentiment score?
- What weighted combination of different aspects best predicts a Reviewer_Score?

2. User Interface:

A user interface is the single most important element that plays a massive role in bringing in high volumes of users. If our tool's user interface can evolve from a command line interface to a user-friendly graphical web browser, more people are encouraged to use it and provide more feedback on how to continually improve its functionality.

## Reference


Adeborna, E., & Siau, K. (2014). An Approach to Sentiment Analysis – The Case of Airline Quality Rating. *Pacific Asia Conference on Information Systems (PACIS) .*

Alaei, A., Becken, S., & Stantic, B. (2017). Sentiment analysis in tourism: Capitalising on Big Data. *Journal of Travel Research*.

Brob, J. (2013). *Aspect-oriented sentiment analysis of customer reviews using distant supervision techniques.* PhD Thesis, Department of Mathematics and Computer Science, University of Berlin.

Hutto, C., & Gilbert, E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. *Association for the Advancement of Artificial Intelligence*.

Liu, B. (2012). *Sentiment Analysis and Opinion Mining.* Synthesis Lectures on Human Language Technologies.

Marrese-Taylora, E., Velasqueza, J. D., Bravo-Marquezb, F., & Matsuoc, Y. (2013). Identifying Customer Preferences about Tourism Products using an Aspect-Based Opinion Mining Approach. *17th International Conference in Knowledge Based and Intelligent Information and Engineering Systems - KES2013*, 182 - 191.

PAurchana.P, PIyyappan.R, & PPeriyasamy.P. (2014). Sentiment Analysis in Tourism. *IJISET - International Journal of Innovative Science, Engineering & Technology*, Vol. 1 Issue 9.

Shi, H.-X., & Li, X.-J. (2011). A sentiment analysis model for hotel reviews based on supervised learning. *2011 International Conference on Machine Learning and Cybernetics.*

T, C. C., & Joseph, S. (2014). Aspect based Opinion Mining from Restaurant Reviews. *International Journal of Computer Applications*.

Zhai, C., & Massun, S. (2016). *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining.* New York: Association for Computing Machinery and Morgan & Claypool.