Sentiment Analysis to identify top and bottom 10 cities for tourism

This python notebook contains code which would be used to scrape information on top 90 cities based on population from wiki travel website and use the informattion to select top 10 cities based on sentiment analysis using the NLTK.

The python notebook is divided into the following segments,

- Scraping the required information from the web
- Using NLTK to process the scraped data and to score the corpora as positive/negative.
- Listing down the top 10 cities based on their score.

Importing all required libraries

- urllib3 to interact with the web
- bs4 to scrape the information off the wiki travel website
- pandas to handle data files
- unidecode to decode accented characters in city names to non-accented form for url formation and file naming.

```
In [12]:
```

```
#refrence url (https://github.com/kmaiya/Presidential_Web_Scrape/blob/master/pre
sScrape.py)
#importing required libraries
import urllib3
from bs4 import BeautifulSoup
import pandas as pd
import unidecode
import os

print(os.chdir("/Users/Avinash/Desktop/ecs_coursework/sentiment_top10cities"))
```

```
None
```

Scraping required information from the web

Creating Functions to form urls and scrape wikitravel

Once the required libraries are imported, functions to form urls from the search list is createdwith the base url of wiki travel, to search on the internet. Also in the below code multi word city names are seperated by an '_' (underscore) as expected in the url.

In [13]:

```
#function for for forming lookup urls
def urlform(skey):
    key=skey.readlines()
    for i in key:
            lkey=(unidecode.unidecode(i.rstrip('\n')))
            #converting city names to lower case doesnt seem to work due to chan
ge in capitalization for url
            c=lkey.split()
            sc=len(c)
            url = "https://wikitravel.org/en/"
            #appending each word from list to wikipedia url to search for tags a
nd entries
            if lkey:
                if sc>1:
                    x=0
                    z=1
                    while x<sc:
                        if z<sc:
                            url=url+c[x]+"_"
                        else:
                            url=url+c[x]
                        x=x+1
                        z=z+1
                else:
                    url=url+lkey
                scrape2file(url,lkey)
            else:
                pass
    skey.close()
```

once the urls with the search key (city name) are formed the urllib3 and the bs4 packages are used to connect to ('GET') and extract content from the wiki travel webpages, using specific tags (paragraph,

) where the required information is present. Also the all the information collected from the wiki travel webpages are written onto a text file which are seperated by special character chains with the city name, such as given below.

*********Chennai*********

and end with

#####################

In [14]:

```
#function for writing the scraped data to a single file
def scrape2file(url,lkey):
    http=urllib3.PoolManager()
    cont=http.request("GET",url)
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
    bs=BeautifulSoup(cont.data,'lxml')
    text=bs.find_all('p')
    k=""
    for i in text:
        k=(k+(i.text)+"\n")
    score(stop_remove(lemmatize(tokenize(k.lower()))),lkey)
    file=open("scrape/scraped_cities.txt","a")
    file.writelines([("\n\n"),("*"*30+(lkey).upper()+"*"*30),k,("#"*50),("\n\n\n
\n")])
    file.close()
```

Importing and processing input data (list of cities)

In the below code block the input file from the internet is downloaded and converted to a text file (topcity.txt), (since the function was created and tested for text files). Then the 'urlform' function is called to return the data scraped from wiki travel in a text file (scraped_cities.txt)

Processing the scraped data using NLTK

Tokenizing and normalizing the words from the data scraped for each individual city.

Word tokenizing reduces the structure of the data into smaller segments (words) based on specific separation delimiter usually whitespace. In this case a regular expression to identify word characters and characters follows after (\w) excluding punctuations and other special characters, also the tokenized words are stored in a text file.

Normalization of the text is done by the below stages,

- Removing stop words (which in this work don't mean much, only lower case stop words are considered so convert text to lower case)
- Removing numbers and words that are not present in the english dictionary

Stemming and Lemmatization

The tokenized text is lemmatized to bring the words to their root form, so that it is easier to compare with the lexicon suring sentiment analysis. Also stemming (to remove word suffixes) doesnt seem to produce a reliable list of words so is not considered here.

Removing stop words

Once the lemmatization is done, words of less importance such as is , the, or, at etc. are removed with the similar purpose as lemmatization.

In [84]:

```
def tokenize(input):
    from nltk import RegexpTokenizer
    tokenizer=RegexpTokenizer(r'\w+')
    tokenized=tokenizer.tokenize(input)
    #tok=open("scrape/tokenized.txt","a")
    #tok.write(str(tokenized))
    return tokenized

def lemmatize(input):
    from nltk.stem import WordNetLemmatizer
    lemm=WordNetLemmatizer()
    lemmat=[lemm.lemmatize(w) for w in input]
    #lem=open("scrape/lemmatized.txt","a")
    #lem.write(str(lemmat))
    #lem.close()
    return lemmat

#def stemmer(input):
#    from nltk.stem.porter import PorterStemmer
#    stemer=PorterStemmer()
#    stem=[stemer.stem(w) for w in input]
#    st=open("stemmed_text.txt",'a')
#    st.write(str(stem))
#    st.close()

def stop_remove(input):
    from nltk.corpus import stopwords
    stop=set(stopwords.words('english'))
    norm_stop=[w for w in input if not w in stop]
    #stopwrd=open("scrape/stop_removed.txt","a")
    #stopwrd.write(str(norm_stop))
    #stopwrd.close()
    return norm_stop

def score(input,cit):
    from nltk.sentiment.vader import  SentimentIntensityAnalyzer
    sent=SentimentIntensityAnalyzer()
    comp=0
    for w in input:
        senti=sent.polarity_scores(w)
        comp=comp+senti['compound']
    #if condition below prevents errors due to division
    # by zero sometimes when empty lists are created from data
    if len(input)>0:
        avgcomp=comp/len(input)
        sentiment=open("scrape/sentiment_text.csv","a")
        sentiment.writelines([str(cit),",",str(avgcomp),("\n")])
        sentiment.close()
        #below doesnt work df reinitialized on each iteration
        #df=[]
        #df.append({"City":str(cit),"Score":avgcomp})
        #print(df)
        #df=pd.DataFrame(columns=["City","Score"])
        #df.loc[cit]=[str(cit) , avgcomp]
        #df.to_csv("df.csv", sep=",")
    else:
        pass
```

Text from two sources are used in this work using the below two cells. One, by importing city names and scraping relevant information from the web, and the other by reading from a text file containing data that is already scraped.

In [64]:

```
#from scraped text
#importing list of cities from url to generate wiki urls
cities=pd.read_csv("http://www.downloadexcelfiles.com/sites/default/files/docs/l
ist-worlds-largest-cities-707j.csv", encoding='latin-1')
cities['City'].to_csv("scrape/topcity.txt", index=False)
cities=open("scrape/topcity.txt","r")
urlform(cities)
```

When importing data from the text file, the text is split and the city names are identified using a regular expression and is processed for sentiment analysis.

In [20]:

```
#from file
import re
f = open("city_wiki_details.txt", 'r')
data = f.read()
split=data.split("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")
for i in split:
    pattern='^.[A-Za-z][^x]*'
    match=re.findall(pattern,i)
    cit=""
    for place in match: cit=place
    score(stop_remove(lemmatize(tokenize(i))),cit)
```

The below code creates two text files with the top 10 best cities and bottom 10 for further analysis.

In [83]:

```
#print top  10 cities
scores=pd.read_csv("scrape/sentiment_text.csv", header=None)
scores.columns=['City','Score']
sort_top=scores.sort_values(by=['Score'], ascending=[False])
sort_bottom=scores.sort_values(by=['Score'], ascending=[True])
sort_top.head(10)
top=open("top_10.txt",'w')
top.write(str(sort_top.head(10)))
top.close()
bot=open("bot_10.txt",'w')
bot.write(str(sort_bottom.head(10)))
bot.close()
```

Conclusion

Although the above code produces the top 10 and the bottom 10 cities for tourism, it is far from accurate and can be improved by using other methods, maybe by using a better lexicon more relevant to the tourism industry and better input data and data from multiple sources combined and normalized.