COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Week 10

Text Mining — Automatic Understanding of Text

Applied Data Science

Columbia University - Columbia Engineering

Course Agenda

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

- Week 1: Python Basics: How to Translate Procedures into Codes
- Week 2: Intermediate Python — Data structures for Your Analysis
- Week 3: Relational Databases — Where Big Data is Typically Stored
- Week 4: SQL — Ubiquitous Database Format/Language
- Week 5: Statistical Distributions — The Shape of Data
- Week 6: Sampling — When You Can't or Won't Have ALL the Data
- Week 7:Hypothesis Testing — Answering Questions about Your Data
- Week 8: Data Analysis and Visualization — Using Python's NumPy for Analysis
- Week 9: Data analysis and visualization — Using Python's Pandas for Data Wrangling
- **Week 10: Text Mining — Automatic Understanding of Text**
- Week 11: Machine learning — Basic Regression and Classification
- Week 12: Machine learning — Decision Trees and Clustering

Nlkt setup

```
In [*]: import nltk
        nltk.download()

        showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

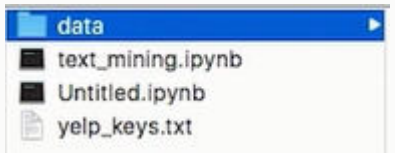
Packages will start downloading in the background

Identifier	Name	Size	Status
all	All packages	n/a	installed
all-corpora	All the corpora	n/a	installed
all-nltk	All packages available on nltk_data gh-pages branch	n/a	installed
book	Everything used in the NLTK Book	n/a	installed
popular	Popular packages	n/a	installed
third-party	Third-party data packages	n/a	installed

Output should show “true” once it’s downloaded

```
Out[1]: True
```

Data that needs to be imported



So, these two files you need to move from this location into the location where we downloaded all the 'nltk' download stuff.

Showing info

https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.html

```
In [*]: import nltk
        nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

Working with text!

Sentiment Analysis

Identify entities and emotions in a sentence and use these to determine if the entity is being viewed positively or negatively

Easy examples

- I had an **excellent** souffle at the restaurant Cavity Maker
- Excellent is a positive word for both the souffle as well as for the restaurant

Not so easy examples

Often, looking at words alone is not enough to figure out the sentiment

- *The Girl on the Train is an excellent book for a 'stuck at home' snow day*

This one is easy since it includes an explicit positive opinion using a positive word

Showing info

https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.html

Not so easy examples

Often, looking at words alone is not enough to figure out the sentiment

- *The Girl on the Train is an excellent book for a 'stuck at home' snow day*

This one is easy since it includes an explicit positive opinion using a positive word

- *The Girl on the Train is an excellent book for using as a liner for your cat's litter box*

Not so simple! The positive word "excellent" is used with a negative connotation.

- *The Girl on the Train is better than Gone Girl*

The positive word is used as a comparator. Whether the writer likes The Girl on the Train or not depends on what he or she thinks of Gone Girl

Sentiment analysis is generally a starting point in analyzing a text and is then coupled with other techniques (e.g., topic analysis)

Sentiment analysis is usually done using a corpus of positive and negative words

- Some sources compile lists of positive and negative words
- Others include the polarity - the degree of positivity or negativity - of each word

Sources for Sentiment-Coded Words

NRG Emotion Lexicon: words coded into emotional categories (many languages)

http://ptrckprry.com/course/ssd/data/positive-words.txt
http://ptrckprry.com/course/ssd/data/negative-words.txt

NRG Emotion Lexicon: words coded into emotional categories (many languages)

http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm

SentiWordNet: List of words weighted by positive or negative sentiments. Includes guidance on how to use the words

http://sentiwordnet.isti.cnr.it/

Vadar Sentiment tool: 7800 words with positive and negative polarity

included with python nltk

Our examples

- Compiled set of 15 reviews each of four neighborhood restaurants
- Presidential inaugural addresses (from Washington to Trump)
- Some data from yelp (very limited!)

Simple Sentiment Analysis


```
]: def get_words(url):
       import requests
       words = requests.get(url).content.decode('latin-1')
       word_list = words.split('\n')
       index = 0
       while index < len(word_list):
           word = word_list[index]
           if ';' in word or not word:
               word_list.pop(index)
           else:
               index+=1
       return word_list

   #Get lists of positive and negative words
   p_url = 'http://ptrckprry.com/course/ssd/data/positive-words.txt'
   n_url = 'http://ptrckprry.com/course/ssd/data/negative-words.txt'
   positive_words = get_words(p_url)
   negative_words = get_words(n_url)
```

```
In [3]: len(positive_words)

Out[3]: 2006
```

```
;
; Opinion Lexicon: Positive
;
; This file contains a list of POSITIVE opinion words (or sentiment words
;
; This file and the papers can all be downloaded from
;    http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html
;
; If you use this list, please cite one of the following two papers:
;
;   Minqing Hu and Bing Liu. "Mining and Summarizing Customer Reviews."
;       Proceedings of the ACM SIGKDD International Conference on Knowled
;       Discovery and Data Mining (KDD-2004), Aug 22-25, 2004, Seattle,
;       Washington, USA,
;   Bing Liu, Minqing Hu and Junsheng Cheng. "Opinion Observer: Analyzing
;       and Comparing Opinions on the Web." Proceedings of the 14th
;       International World Wide Web conference (WWW-2005), May 10-14,
;       2005, Chiba, Japan.
;
; Notes:
;    1. The appearance of an opinion word in a sentence does not necessar
;       mean that the sentence expresses a positive or negative opinion.
;       See the paper below:
;
;       Bing Liu. "Sentiment Analysis and Subjectivity." An chapter in
;          Handbook of Natural Language Processing, Second Edition,
;          (editors: N. Indurkhya and F. J. Damerau), 2010.
;
;    2. You will notice many misspelled words in the list. They are not
;       mistakes. They are included as these misspelled words appear
;       frequently in social media content.
;
```

Simple Sentiment Analysis

Compute sentiment by looking at the proportion of positive and negative words in the text

In [4]:
```
from nltk import word_tokenize
cpos = cneg = lpos = lneg = 0
for word in word_tokenize(community):
    if word in positive_words:
        cpos+=1
    if word in negative_words:
        cneg+=1
for word in word_tokenize(le_monde):
    if word in positive_words:
        lpos+=1
    if word in negative_words:
        lneg+=1
print("community {0:1.2f}%\t {1:1.2f}%\t {2:1.2f}%".format(cpos/len(word_tokenize(community))*100,
                                                          cneg/len(word_tokenize(community))*100,
                                                          (cpos-cneg)/len(word_tokenize(community))*100))
print("le monde  {0:1.2f}%\t {1:1.2f}%\t {2:1.2f}%".format(lpos/len(word_tokenize(le_monde))*100,
                                                          lneg/len(word_tokenize(le_monde))*100,
                                                          (lpos-lneg)/len(word_tokenize(le_monde))*100))
```

```
community 5.09%   1.12%    3.97%
le monde  5.33%   1.49%    3.85%
```

Simple Sentiment Analysis Using NRC Data

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Simple sentiment analysis using NRC data

- NRC data codifies words with emotions
- 14,182 words are coded into 2 sentiments and 8 emotions

For example, the word abandonment is associated with anger, fear, sadness and has a negative sentiment

- abandoned anger 1
- abandoned anticipation 0
- abandoned disgust 0
- abandoned fear 1
- abandoned joy 0
- abandoned negative 1
- abandoned positive 0
- abandoned sadness 1
- abandoned surprise 0
- abandoned trust 0

Read the NRC sentiment data

In [6]:
```
nrc = "data/NRC-emotion-lexicon-wordlevel-alphabetized-v0.92.txt"
count=0
emotion_dict=dict()
with open(nrc,'r') as f:
    all_lines = list()
    for line in f:
        if count < 46:
            count+=1
            continue
        line = line.strip().split('\t')
        if int(line[2]) == 1:
            if emotion_dict.get(line[0]):
                emotion_dict[line[0]].append(line[1])
            else:
                emotion_dict[line[0]] = [line[1]]
```

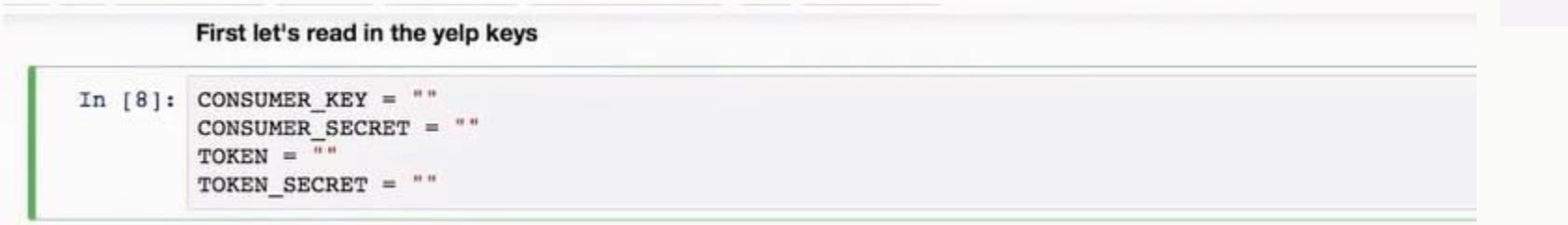
In [7]:
```
emotion_dict
```

Out[7]:
```
{'abacus': ['trust'],
 'abandon': ['fear', 'negative', 'sadness'],
 'abandoned': ['anger', 'fear', 'negative', 'sadness'],
 'abandonment': ['anger', 'fear', 'negative', 'sadness', 'surprise'],
 'abba': ['positive'],
 'abbot': ['trust'],
 'abduction': ['fear', 'negative', 'sadness', 'surprise'],
 'aberrant': ['negative'],
 'aberration': ['disgust', 'negative'],
```

Caveat: We're only looking at one review snippet for each restaurant

1. Download yelp python "pip install yelp"
2. Register with yelp https://www.yelp.com/developers/manage_apikeys (use anything for the host)
3. Copy the various keys into variables as below

We'll see what we can figure out from reviews of restaurants close to Columbia



Steps to Follow

We need to do a few things

- Get the latitude and longitude of our location
- Set up the parameters for what data we want from yelp
- Query yelp by passing authentication info as well as our parameters
- Extract review snippets

In [11]:
```
#We'll use the get_lat_lng function we wrote way back in week 3
def get_lat_lng(address):
    url = 'https://maps.googleapis.com/maps/api/geocode/json?address='
    url += address
    import requests
    response = requests.get(url)
    if not (response.status_code == 200):
        return None
    data = response.json()
    if not( data['status'] == 'OK'):
        return None
    main_result = data['results'][0]
    geometry = main_result['geometry']
    latitude = geometry['location']['lat']
    longitude = geometry['location']['lng']
    return latitude,longitude
```

In [11]:
```
lat,long = get_lat_lng("Columbia University")
```

In [12]:
```
#Now set up our search parameters
def set_search_parameters(lat,long,radius):
  #See the Yelp API for more details
    params = {}
    params["term"] = "restaurant"
    params["ll"] = "{},{}".format(str(lat),str(long))
    params["radius_filter"] = str(radius) #The distance around our point in metres
    params["limit"] = "10" #Limit ourselves to 10 results

    return params
```

In [13]:
```
set_search_parameters(lat,long,200)
```

Out[13]:
```
{'limit': '10',
 'll': '40.8075355,-73.9625727',
 'radius_filter': '200',
 'term': 'restaurant'}
```

Rauth Library to Handle Authentications


```
In [*]: !pip install rauth

Collecting rauth
  Downloading rauth-0.7.3.tar.gz
Requirement already satisfied: requ
uth)
Requirement already satisfied: cert
requests>=1.2.3->rauth)
```

```
In [19]: def get_results(params):
             import rauth
             consumer_key = CONSUMER_KEY
             consumer_secret = CONSUMER_SECRET
             token = TOKEN
             token_secret = TOKEN_SECRET

             session = rauth.OAuth1Session(
             consumer_key = consumer_key
             ,consumer_secret = consumer_secret
             ,access_token = token
             ,access_token_secret = token_secret)

             request = session.get("http://api.yelp.com/v2/search",params=params)
             #print(request.url)
             #Transforms the JSON API response into a Python dictionary
             data = request.json()
             session.close()

             return data
```

```
In [20]: #Get the results
         response = get_results(set_search_parameters(get_lat_lng("Community Food and Juice")[0],get_lat_lng("Community Food and Juice"
```

```
In [21]: response

Out[21]: {'businesses': [{'categories': [['Cajun/Creole', 'cajun']],
             'display_phone': '+1-646-861-2152',
             'distance': 97.75276677876506,
             'id': 'shaking-crab-new-york-4',
             'image_url': 'https://s3-media1.fl.yelpcdn.com/bphoto/pbe8R3mrdtVsNGQKVN7EJQ/ms.jpg',
             'is_claimed': True,
             'is_closed': False,
             'location': {'address': ['2869 Broadway'],
              'city': 'New York',
              'coordinate': {'latitude': 40.80522, 'longitude': -73.96639},
```

Rauth Library to Handle Authentications

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Extracting snippets

```
In [26]: response['businesses'][0]['id']

Out[26]: 'shaking-crab-new-york-4'

In [27]: all_snippets = list()
         for business in response['businesses']:
             name = business['name']
             snippet = business['snippet_text']
             id_no = business['id']
             all_snippets.append((id_no,name,snippet))
         all_snippets

Out[27]: [('shaking-crab-new-york-4',
           'Shaking Crab',
           'Easily the best and freshest Cajun-style crawdads ever!
         s langostinos.. \n\nI asked for the hottest...'),
          ('community-food-and-juice-new-york',
           'Community Food & Juice',
           'Was brought here by some friends/regulars who live in t
         or! It was crispy and fluffy and light and...'),
          ('dig-inn-new-york-14',
           'Dig Inn',
           "Delicious!  It's a great alternative to fast food; grea
         ions are pretty large, and the seasonings..."),
```

Functionalize this

```
In [28]: def get_snippets(response):
             all_snippets = list()
             for business in response['businesses']:
                 name = business['name']
                 snippet = business['snippet_text']
                 id = business['id']
                 all_snippets.append((id,name,snippet))
             return all_snippets
```

Functions Analyzing Emotions

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

```
In [ ]: def emotion_analyzer(text,emotion_dict=emotion_dict):
            #Set up the result dictionary
            emotions = {x for y in emotion_dict.values() for x in y}
            emotion_count = dict()
            for emotion in emotions:
                emotion_count[emotion] = 0

            #Analyze the text and normalize by total number of words
            total_words = len(text.split())
            for word in text.split():
                if emotion_dict.get(word):
                    for emotion in emotion_dict.get(word):
                        emotion_count[emotion] += 1/len(text.split())
            return emotion_count
```

We can analyze the emotional content of the review snippet

```
In [33]: print("%-12s %1s\t%1s %1s %1s %1s   %1s %1s %1s %1s"%(
                "restaurant","fear","trust","negative","positive","joy","disgust","anticip",
                "sadness","surprise"))

         for snippet in all_snippets:
             text = snippet[2]
             result = emotion_analyzer(text)
             print("%-12s %1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f"%(
                 snippet[1][0:10],result['fear'],result['trust'],
                   result['negative'],result['positive'],result['joy'],result['disgust'],
                   result['anticipation'],result['sadness'],result['surprise']))
```

restaurant	fear	trust	negative	positive	joy	disgust	anticip	sadness	surprise
Shaking Cr	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Community	0.04	0.00	0.04	0.00	0.00	0.00	0.00	0.04	0.00
Dig Inn	0.00	0.04	0.00	0.04	0.04	0.00	0.04	0.00	0.00
Le Monde	0.00	0.07	0.04	0.04	0.04	0.00	0.00	0.00	0.00
Mill Korea	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00
Symposium	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Tom's Rest	0.00	0.14	0.03	0.24	0.14	0.00	0.10	0.00	0.00
Mel's Burg	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00
Amigos	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00
The Height	0.00	0.11	0.00	0.07	0.04	0.00	0.07	0.00	0.00

Functions Analyzing Emotions

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

In [34]:
```
def comparative_emotion_analyzer(text_tuples):
    print("%-20s %ls\t%ls %ls %ls %ls    %ls %ls %ls %ls"%(
            "restaurant","fear","trust","negative","positive","joy","disgust","anticip",
            "sadness","surprise"))

    for text_tuple in text_tuples:
        text = text_tuple[2]
        result = emotion_analyzer(text)
        print("%-20s %1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f"%(
            text_tuple[1][0:20],result['fear'],result['trust'],
              result['negative'],result['positive'],result['joy'],result['disgust'],
              result['anticipation'],result['sadness'],result['surprise']))

#And test it
comparative_emotion_analyzer(all_snippets)
```

restaurant	fear	trust	negative	positive	joy	disgust	anticip	sadness	surprise
Shaking Crab	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Community Food & Jui	0.04	0.00	0.04	0.00	0.00	0.00	0.00	0.04	0.00
Dig Inn	0.00	0.04	0.00	0.04	0.04	0.00	0.04	0.00	0.00
Le Monde	0.00	0.07	0.04	0.04	0.04	0.00	0.00	0.00	0.00
Mill Korean	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00
Symposium Greek Rest	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Tom's Restaurant	0.00	0.14	0.03	0.24	0.14	0.00	0.10	0.00	0.00
Mel's Burger Bar	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00
Amigos	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00
The Heights Bar & Gr	0.00	0.11	0.00	0.07	0.04	0.00	0.07	0.00	0.00

Functionalize with the yelp

In [35]:
```
def analyze_nearby_restaurants(address,radius):
    lat,long = get_lat_lng(address)
    params = set_search_parameters(lat,long,radius)
    response = get_results(params)
    snippets = get_snippets(response)
    comparative_emotion_analyzer(snippets)

#And test it
analyze_nearby_restaurants("Community Food and Juice",200)
```

restaurant	fear	trust	negative	positive	joy	disgust	anticip	sadness
Shaking Crab	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Community Food & Jui	0.04	0.00	0.04	0.00	0.00	0.00	0.00	0.04
Dig Inn	0.00	0.04	0.00	0.04	0.04	0.00	0.04	0.00
Le Monde	0.00	0.07	0.04	0.04	0.04	0.00	0.00	0.00
Mill Korean	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00
Symposium Greek Rest	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Tom's Restaurant	0.00	0.14	0.03	0.24	0.14	0.00	0.10	0.00
Mel's Burger Bar	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00
Amigos	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00
The Heights Bar & Gr	0.00	0.11	0.00	0.07	0.04	0.00	0.07	0.00

In [36]:
```
#Test it on some other place
analyze_nearby_restaurants("221 Baker Street",200)
```

restaurant	fear	trust	negative	positive	joy	disgust	anticip	sadness
Opa	0.00	0.03	0.00	0.03	0.03	0.00	0.03	0.00
Beast + Barrel	0.05	0.15	0.10	0.25	0.15	0.05	0.25	0.05
Brewhaus	0.00	0.04	0.00	0.04	0.04	0.00	0.04	0.00
Easy Bistro & Bar	0.00	0.06	0.00	0.06	0.06	0.00	0.06	0.03
Aretha Frankensteins	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00
Milk & Honey	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Tony's Pasta Shop an	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00

Simple Analysis: Word Clouds

Let's see what sort of words the snippets use

- First we'll combine all snippets into one string
- Then we'll generate a word cloud using the words in the string
- You may need to install wordcloud using pip
- pip install wordcloud

In [ ]:
```
all_snippets
```

In [ ]:
```
text=''
for snippet in all_snippets:
    text+=snippet[2]
text
```

Simple Analysis: Word Clouds

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

In [40]:
```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
%matplotlib inline

wordcloud = WordCloud(stopwords=STOPWORDS,background_color='white',width=3000,height=3000).generate(text)


plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```

Detailed Analysis

Let's do a detailed comparison of local restaurants

I've saved a few reviews for each restaurant in four directories

We'll use the PlainTextCorpusReader to read these directories

- PlainTextCorpusReader reads all matching files in a directory and saves them by file-ids

In [ ]:
```
import nltk
from nltk.corpus import PlaintextCorpusReader
community_root = "data/community"
le_monde_root = "data/le_monde"
community_files = "community.*"
le_monde_files = "le_monde.*"
heights_root = "data/heights"
heights_files = "heights.*"
amigos_root = "data/amigos"
amigos_files = "amigos.*"
community_data = PlaintextCorpusReader(community_root,community_files)
le_monde_data = PlaintextCorpusReader(le_monde_root,le_monde_files)
heights_data = PlaintextCorpusReader(heights_root,heights_files)
amigos_data = PlaintextCorpusReader(amigos_root,amigos_files)
```

In [42]:
```
amigos_data.fileids()
```

Out[42]:
```
['amigos.1',
 'amigos.10',
 'amigos.11',
 'amigos.12',
 'amigos.13',
 'amigos.14',
 'amigos.15',
 'amigos.16',
 'amigos.17',
 'amigos.18',
 'amigos.19',
 'amigos.2',
 'amigos.20',
 'amigos.21',
 'amigos.3',
 'amigos.4',
 'amigos.5',
 'amigos.6',
 'amigos.7',
```

In [43]:
```
amigos_data.raw()
```

Out[43]:
```
'I see all these bad reviews, but speaking for r
nce in the least--and I\'m here about once a wee
've never had a bad dish, although some are mere
, and the margaritas are among the best I\'ve ev
at specials during the week--I believe one day i
t sounds so impossible even as I type it but I\'
inco de Mayo but we wanted something close to ho
end telling me about this place a while back. So
that it was packed BUT the receptionist explaine
\n\nWe found some seats at the bar and got some
reat,  the food was great,  the drinks were AMAZ
FINITELY be going back with even more people! I
```

Detailed Analysis

We need to modify comparitive_emotion_analyzer to tell it where the restaurant name and the text is in the tuple

In [ ]:
```
def comparative_emotion_analyzer(text_tuples,name_location=1,text_location=2):
    print("%-20s %1s\t%1s %1s %1s %1s   %1s %1s %1s %1s"%(
            "restaurant","fear","trust","negative","positive","joy","disgust","anticip",
            "sadness","surprise"))

    for text_tuple in text_tuples:
        text = text_tuple[text_location]
        result = emotion_analyzer(text)
        print("%-20s %1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f\t%1.2f"%(
            text_tuple[name_location][0:20],result['fear'],result['trust'],
              result['negative'],result['positive'],result['joy'],result['disgust'],
              result['anticipation'],result['sadness'],result['surprise']))

#And test it
comparative_emotion_analyzer(all_snippets)
```

restaurant	fear	trust	negative	positive	joy	disgust	anticip	sadness	surprise
Shaking Crab	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Community Food & Jui	0.04	0.00	0.04	0.00	0.00	0.00	0.00	0.04	0.00
Dig Inn	0.00	0.04	0.00	0.04	0.04	0.00	0.04	0.00	0.00
Le Monde	0.00	0.07	0.04	0.04	0.04	0.00	0.00	0.00	0.00
Mill Korean	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00
Symposium Greek Rest	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Tom's Restaurant	0.00	0.14	0.03	0.24	0.14	0.00	0.10	0.00	0.00
Mel's Burger Bar	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00
Amigos	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00
The Heights Bar & Gr	0.00	0.11	0.00	0.07	0.04	0.00	0.07	0.00	0.00

In [45]:
```
restaurant_data = [('community',community_data.raw()),('le monde',le_monde_data.raw())
                   ,('heights',heights_data.raw()), ('amigos',amigos_data.raw())]
comparative_emotion_analyzer(restaurant_data,0,1)
```

restaurant	fear	trust	negative	positive	joy	disgust	anticip	sadness	surprise
community	0.00	0.03	0.01	0.05	0.03	0.01	0.02	0.01	0.01
le monde	0.00	0.03	0.01	0.04	0.02	0.00	0.02	0.00	0.01
heights	0.00	0.03	0.01	0.04	0.03	0.00	0.03	0.01	0.01
amigos	0.01	0.03	0.01	0.04	0.03	0.01	0.02	0.01	0.01

Simple Analysis: Complexity

We'll look at four complexity factors

- average word length: longer words adds to complexity
- average sentence length: longer sentences are more complex (unless the text is rambling!)
- vocabulary: the ratio of unique words used to the total number of words (more variety, more complexity)

token: A sequence (or group) of characters of interest. For e.g., in the below analysis, a token = a word

- Generally: A token is the base unit of analysis
- So, the first step is to convert text into tokens and nltk text object

Simple Analysis: Complexity

In [46]:
```
#Construct tokens (words/sentences) from the text
text = le_monde_data.raw()
import nltk
from nltk import sent_tokenize,word_tokenize
sentences = nltk.Text(sent_tokenize(text))
print(len(sentences))
words = nltk.Text(word_tokenize(text))
print(len(words))
```

```
188
2595
```

In [47]:
```
num_chars=len(text)
num_words=len(word_tokenize(text))
num_sentences=len(sent_tokenize(text))
vocab = {x.lower() for x in word_tokenize(text)}
print(num_chars,int(num_chars/num_words),int(num_words/num_sentences),(len(vocab)/num_words))
```

```
12332 4 13 0.29132947976878615
```

Let's functionalize this

In [48]:
```
def get_complexity(text):
    num_chars=len(text)
    num_words=len(word_tokenize(text))
    num_sentences=len(sent_tokenize(text))
    vocab = {x.lower() for x in word_tokenize(text)}
    return len(vocab),int(num_chars/num_words),int(num_words/num_sentences),len(vocab)/num_words
```

In [49]:
```
get_complexity(le_monde_data.raw())
```

Out[49]:
```
(756, 4, 13, 0.29132947976878615)
```

In [51]:
```
for text in restaurant_data:
    (vocab,word_size,sent_size,vocab_to_text) = get_complexity(text[1])
    print("{0:15s}\t{1:1.2f}\t{2:1.2f}\t{3:1.2f}\t{4:1.2f}".format(text[0],vocab,word_size,sent_size,vocab_to_text))
```

```
community       1029.00 4.00    16.00   0.28
le monde        756.00  4.00    13.00   0.29
heights         720.00  4.00    16.00   0.28
amigos          792.00  4.00    15.00   0.24
```

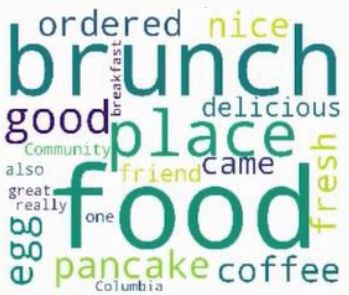
Simple Analysis: Word Cloud Comparison

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

In [52]:
```
texts = restaurant_data
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
%matplotlib inline
#Remove unwanted words
#As we look at the cloud, we can get rid of words that don't make sense by adding them to this variable
DELETE_WORDS = []
def remove_words(text_string,DELETE_WORDS=DELETE_WORDS):
    for word in DELETE_WORDS:
        text_string = text_string.replace(word,' ')
    return text_string

#Remove short words
MIN_LENGTH = 0
def remove_short_words(text_string,min_length = MIN_LENGTH):
    word_list = text_string.split()
    for word in word_list:
        if len(word) < min_length:
            text_string = text_string.replace(' '+word+' ',' ',1)
    return text_string


#Set up side by side clouds
COL_NUM = 2
ROW_NUM = 2
fig, axes = plt.subplots(ROW_NUM, COL_NUM, figsize=(12,12))

for i in range(0,len(texts)):
    text_string = remove_words(texts[i][1])
    text_string = remove_short_words(text_string)
    ax = axes[i%2]
    ax = axes[i//2, i%2] #Use this if ROW_NUM >=2
    ax.set_title(texts[i][0])
    wordcloud = WordCloud(stopwords=STOPWORDS,background_color='white',width=1200,height=1000,max_words=20).generate(te
    ax.imshow(wordcloud)
    ax.axis('off')
plt.show()
```

output

Community



Le monde

heights

Amigos

Each restaurant can be analyzed based on the word clouds

Ntlk: Python's natural language toolkit

Comparing complexity of restaurant reviews won't get us anything useful. Let's look at something more useful

Ntlk documentation link:

http://www.nlkt.org/api/nlkt.html

Commands cheat sheet:

https://blogs.princeton.edu/etc/files/2014/03/Text-Analysis-with NLTK-Cheatsheet.pdf

Ntlk book

http://www.nltk.org/book

nltk contains a large corpora of pre-tokenized text

Load it using the command:

```
nltk.download()
```

Ntlk: Python's natural language toolkit

Often, a comparitive analysis helps us understand text better

Let's look at US President inaugural speeches

Copy the files 2013-Obama.txt and 2017-Trump.txt to the nltk_data/corpora/inaugural directory. nltk_data should be under your home directory

```
In [55]: inaugural.fileids()
```

```
Out[55]: ['1789-Washington.txt',
          '1793-Washington.txt',
          '1797-Adams.txt',
          '1801-Jefferson.txt',
          '1805-Jefferson.txt',
          '1809-Madison.txt',
          '1813-Madison.txt',
          '1817-Monroe.txt',
          '1821-Monroe.txt',
          '1825-Adams.txt',
```

```
In [56]: inaugural.raw('1861-Lincoln.txt')
```

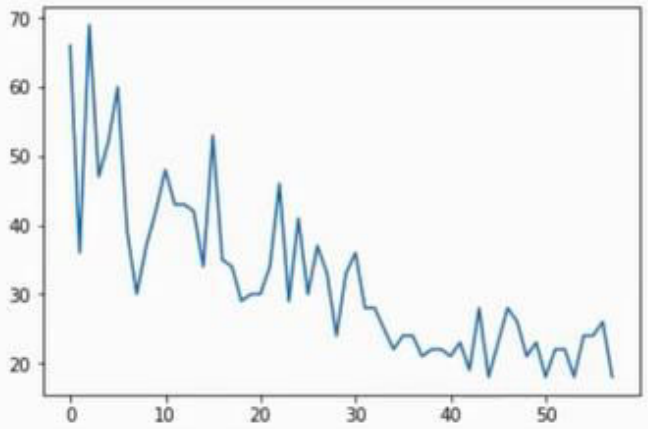
```
Out[56]: 'Fellow-Citizens of the United Stat
         ou to address you briefly and to ta
         o be taken by the President "before
         present for me to discuss those mat
         pprehension seems to exist among th
```

```
In [57]: texts = [('trump',inaugural.raw('2017-Trump.txt')),
                  ('obama',inaugural.raw('2009-Obama.txt')+inaugural.raw('2013-Obama.txt')),
                  ('jackson',inaugural.raw('1829-Jackson.txt')+inaugural.raw('1833-Jackson.txt')),
                  ('washington',inaugural.raw('1789-Washington.txt')+inaugural.raw('1793-Washington.txt'))]
         for text in texts:
             (vocab,word_size,sent_size,vocab_to_text) = get_complexity(text[1])
             print("{0:15s}\t{1:1.2f}\t{2:1.2f}\t{3:1.2f}\t{4:1.2f}".format(text[0],vocab,word_size,sent_size,vocab to text))
```

```
trump           540.00  5.00    18.00   0.32
obama           1349.00 5.00    25.00   0.27
jackson         813.00  5.00    45.00   0.33
washington      636.00  5.00    62.00   0.38
```

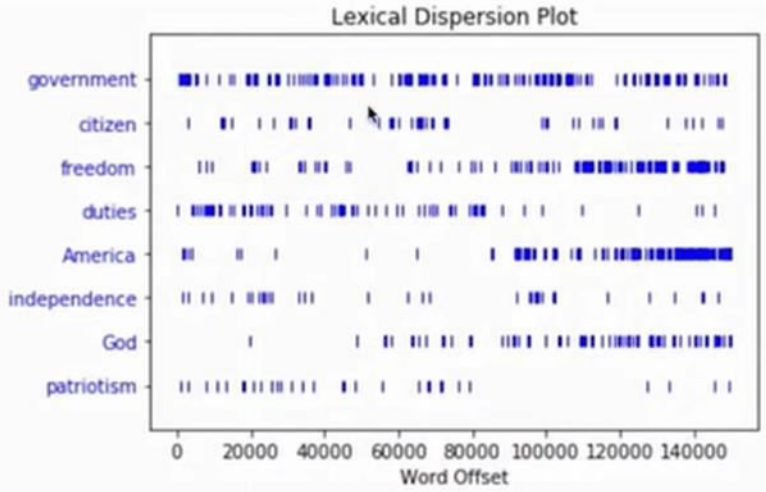
```
In [58]: from nltk.corpus import inaugural
         sentence_lengths = list()
         for fileid in inaugural.fileids():
             sentence_lengths.append(get_complexity(' '.join(inaugural.words(fileid)))[2])
         plt.plot(sentence_lengths)
```

```
Out[58]: [<matplotlib.lines.Line2D at 0x1286352b0>]
```



Digression Plot


```
In [60]: text4.dispersion_plot(["government", "citizen", "freedom", "duties", "America",'independence','God','patriotism'])
```



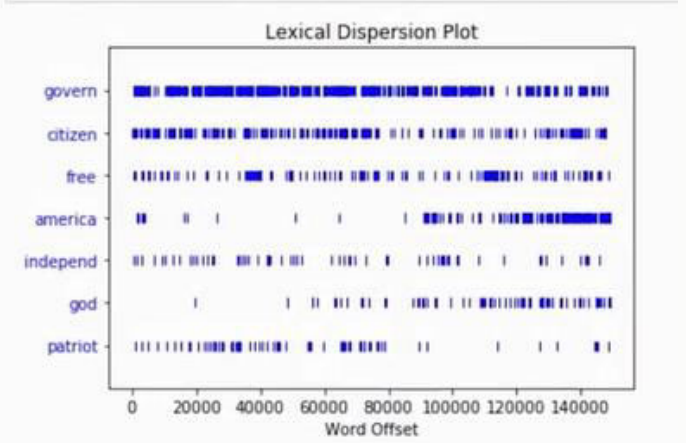
https://tartarus.org/martin/PorterStemmer/

Digression Plot

We may want to use word stems rather than the part of speect form

- For example: patriot, patriotic, patriotism all express roughly the same idea
- nltk has a stemmer that implements the "Porter Stemming Algorithm" https://tartarus.org/martin/PorterStemmer/
- We'll push everything to lowercase as well

In [ ]:
```
from nltk.stem.porter import PorterStemmer
p_stemmer = PorterStemmer()
text = inaugural.raw()
striptext = text.replace('\n\n', ' ')
striptext = striptext.replace('\n', ' ')
sentences = sent_tokenize(striptext)
words = word_tokenize(striptext)
text = nltk.Text([p_stemmer.stem(i).lower() for i in words])
text.dispersion_plot(["govern", "citizen", "free", "america",'independ','god','patriot'])
```



Weighted Word Analysis Using Vader

Vader contains a list of 7500 features weighted by how positive or negative they are

It uses these features to calculate stats on how positive, negative and neutral a passage is

And combines these results to give a compound sentiment (higher = more positive) for the passage

Human trained on twitter data and generally considered good for informal communication

10 humans rated each feature in each tweet in context from -4 to +4

Calculates the sentiment in a sentence using word order analysis

- "marginally good" will get a lower positive score than "extremely good"

Computes a "compound" score based on heuristics (between -1 and +1)

Includes sentiment of emoticons, punctuation, and other 'social media' lexicon elements

```
In [62]: !pip install vaderSentiment
```

```
Collecting vaderSentiment
  Downloading vaderSentiment-2.5-py2.py3-none-any.whl (102kB)
    100% |████████████████████████████████| 112kB 3.7MB/s ta 0:00:01
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-2.5
```

Weighted Word Analysis Using Vader

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

In [63]:
```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

In [ ]:
```
headers = ['pos','neg','neu','compound']
texts = restaurant_data
analyzer = SentimentIntensityAnalyzer()
for i in range(len(texts)):
    name = texts[i][0]
    sentences = sent_tokenize(texts[i][1])
    pos=compound=neu=neg=0
    for sentence in sentences:
        vs = analyzer.polarity_scores(sentence)
        pos+=vs['pos']/(len(sentences))
        compound+=vs['compound']/(len(sentences))
        neu+=vs['neu']/(len(sentences))
        neg+=vs['neg']/(len(sentences))
    print(name,pos,neg,neu,compound)
```

In [65]:
```
restaurant_data
```

Out[65]:
```
[('community',
  'I ate here Monday night for a working dinner with a coworker. I immediately saw its reason for na
ner\'s fandom of the show of the same name: the tablet setup are nearly all communal long tables. Th
w non-shared tables which is fine as not only is NYC filled with such setup but there\'s a livelines
area vibe which prefers this setup. There\'s a shortage of such nice sit-down restaurants in the Col
s place is not only capitalizing on its presence but doing so with great style and flare.\n\nI order
r which was very juicy, perfectly lean, well sauced, cooked as desired (medium-well, for those who w
d with a dill pickle, a handmade cole slaw that lacked mayo (which is good as mayo\'s pure trans-fat
of fries. My coworker got the salmon sandwich, same review applies.\n\nAwesome bar alongside, but we
e it. Vast amounts of seating and an open kitchen where tons of food is prepared live makes for a fu
ccidentally left my headphones, called to ask if it was found, and fetched them the day later. The s
ful, kind, and soft-spoken about recovering and returning it that I\'m duly humbled. Thanks, all!\n\
uite good, and I have no complaints in that area. The reason for only two stars is the cost of that
y overpriced. My wife and I went here for breakfast. We each ordered blueberry pancakes and a cup of
fruit, no juice, no bacon. The pancakes were $13 and the coffee was $4, so the initial bill was $34.
and I left a tip of $6--resulting in a final total of $43 for a breakfast of flapjacks and coffee. N
NY and costs here are higher--but $43!  Ugh!  There are a lot of nice places in the city where you c
d breakfast for 1/3rd to 1/2 less than Community. I recommend you seek them out.\nThis is such a goo
e food, service & ambiance was exactly what we were looking for. We sat at an adorable table on the
```

Weighted Word Analysis Using Vader

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

In [67]:
```
headers = ['pos','neg','neu','compound']
texts = restaurant_data
analyzer = SentimentIntensityAnalyzer()
for i in range(len(texts)):
    name = texts[i][0]
    sentences = sent_tokenize(texts[i][1])
    pos=compound=neu=neg=0
    for sentence in sentences:
        vs = analyzer.polarity_scores(sentence)
        pos+=vs['pos']/(len(sentences))
        compound+=vs['compound']/(len(sentences))
        neu+=vs['neu']/(len(sentences))
        neg+=vs['neg']/(len(sentences))
    print(name,pos,neg,neu,compound)
```

```
community 0.20305855855855845 0.02542342342342343 0.771527027027026 0.3362022522522523
le monde 0.1749627659574468 0.04054255319148937 0.7845053191489363 0.2215510638297871
heights 0.18679354838709677 0.03675483870967742 0.7764387096774197 0.28074129032258055
amigos 0.19447887323943652 0.0497042253521127 0.755830985915493 0.25482488262910796
```

And functionalize this as well

In [68]:
```
def vader_comparison(texts):
    from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
    headers = ['pos','neg','neu','compound']
    print("Name\t",'  pos\t','neg\t','neu\t','compound')
    analyzer = SentimentIntensityAnalyzer()
    for i in range(len(texts)):
        name = texts[i][0]
        sentences = sent_tokenize(texts[i][1])
        pos=compound=neu=neg=0
        for sentence in sentences:
            vs = analyzer.polarity_scores(sentence)
            pos+=vs['pos']/(len(sentences))
            compound+=vs['compound']/(len(sentences))
            neu+=vs['neu']/(len(sentences))
            neg+=vs['neg']/(len(sentences))
        print('%-10s'%name,'%1.2f\t'%pos,'%1.2f\t'%neg,'%1.2f\t'%neu,'%1.2f\t'%compound)
```

In [69]:
```
vader_comparison(restaurant_data)
```

Name	pos	neg	neu	compound
community	0.20	0.03	0.77	0.34
le monde	0.17	0.04	0.78	0.22
heights	0.19	0.04	0.78	0.28
amigos	0.19	0.05	0.76	0.25

Named Entities

People, places, organizations
Named entities are often the subject of sentiments so identifying them can be very useful

Named entity detection is based on Part-of-speech tagging of words and chunks (groups of words)

- Start with sentences (using a sentence tokenizer)
- tokenize words in each sentence
- chunk them. ne_chunk identifies likely chunked candidates (ne = named entity)
- Finally build chunks using nltk's guess on what members of chunk represent (people, place, organization

In [70]:
```
en={}
try:
    sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
    sentences = sent_detector.tokenize(community_data.raw().strip())
    for sentence in sentences:
            tokenized = nltk.word_tokenize(sentence)
            tagged = nltk.pos_tag(tokenized)
            chunked = nltk.ne_chunk(tagged)
            for tree in chunked:
                if hasattr(tree, 'label'):
                    ne = ' '.join(c[0] for c in tree.leaves())
                    en[ne] = [tree.label(), ' '.join(c[1] for c in tree.leaves())]
except Exception as e:
    print(str(e))
import pprint
pp = pprint.PrettyPrinter(indent=4)
pp.pprint(en)
```

```
{   'America': ['GPE', 'NNP'],
    'Awesome': ['GPE', 'NNP'],
    'BEST': ['ORGANIZATION', 'NNP'],
    'Bill': ['PERSON', 'NN'],
    'Boston': ['GPE', 'NNP'],
    'Bottomless': ['GPE', 'NNP'],
    'Broadway': ['GPE', 'NNP'],
    'Brooklyn': ['GPE', 'NNP'],
    'Brunch': ['PERSON', 'NNP'],
    'CU': ['ORGANIZATION', 'NNP'],
    'Came': ['GPE', 'NN'],
```

In [71]:
```
meaningful_sents = list()
i=0
for sentence in sentences:
    if 'service' in sentence:
        i+=1
        meaningful_sents.append((i,sentence))

vader_comparison(meaningful_sents)
```

Name	pos	neg	neu	compound
1	0.00	0.00	1.00	0.00
2	0.11	0.15	0.73	-0.17
3	0.53	0.00	0.47	0.84
4	0.28	0.00	0.71	0.49
5	0.23	0.00	0.78	0.49
6	0.36	0.00	0.64	0.74

Named Entities


```
In [72]: def get_affect(text,word,lower=False):
             import nltk
             from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
             analyzer = SentimentIntensityAnalyzer()
             sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
             sentences = sent_detector.tokenize(text.strip())
             sentence_count = 0
             running_total = 0
             for sentence in sentences:
                 if lower: sentence = sentence.lower()
                 if word in sentence:
                     vs = analyzer.polarity_scores(sentence)
                     running_total += vs['compound']
                     sentence_count += 1
             if sentence_count == 0: return 0
             return running_total/sentence_count
```

```
In [73]: get_affect(community_data.raw(),'service',True)
Out[73]: 0.4321428571428571

In [74]: get_affect(le_monde_data.raw(),'service',True)
Out[74]: 0.22273125
```

The nltk function concordance returns text fragments around a word

```
In [81]: nltk.Text(community_data.words()).concordance('service',100)
Displaying 7 of 7 matches:
 This is such a good brunch spot ! The food , service & ambiance was exactly what we were looking f
 I dug right in due to brunch hunger and slow service . I ' d come back again , but be sure to have
 . Will be exploring more items on the menu . Service is great - our water was also filled , our co
 All in all , a pleasant experience with nice service and nice company . It ' s not every day I get
 place was packed . Even though it was packed service was very good . The beat salad and rice bowl
e with a friend - we were seated promptly and service was attentive . The decor was beautiful and t
h juice n on site coffee . Thanks to the kind service as well , i will remember this restaurant as
```

Text Summarization

Text summarization is useful because you can generate a short summary of a large piece of text automatically. Then, these summaries can serve as an input into a topic analyzer to figure out what the main topic of the text is

A naïve form of summary is to identify the most frequent words in a piece of text and use the occurrence of these words in sentence to rate the importance of a sentence.

```
In [ ]: from nltk.tokenize import word_tokenize
        from nltk.tokenize import sent_tokenize
        from nltk.probability import FreqDist
        from nltk.corpus import stopwords
        from collections import OrderedDict
        import pprint
```

Then prep the text. Get did of end of line chars

```
In [ ]: text = community_data.raw()
        summary_sentences = []
        candidate_sentences = {}
        candidate_sentence_counts = {}
        striptext = text.replace('\n\n', ' ')
        striptext = striptext.replace('\n', ' ')
```

Construct a list of words after getting rid of unimportant ones and numbers

```
In [ ]: words = word_tokenize(striptext)
        lowercase_words = [word.lower() for word in words
                           if word not in stopwords.words() and word.isalpha()]
```

Construct word frequencies and choose the most common n (20)

```
word_frequencies = FreqDist(lowercase_words)
most_frequent_words = FreqDist(lowercase_words).most_common(20)
pp = pprint.PrettyPrinter(indent=4)
pp.pprint(most_frequent_words)
```

```
[   ('i', 73),
    ('the', 46),
    ('food', 20),
    ('brunch', 20),
    ('good', 19),
    ('place', 15),
    ('nice', 13),
    ('ordered', 13),
    ('pancakes', 13),
    ('fresh', 13),
    ('came', 13),
    ('my', 12),
    ('coffee', 12),
    ('delicious', 12),
    ('we', 10),
    ('community', 10),
```

Text Summarization

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

```
In [93]: sentences = sent_tokenize(striptext)
         for sentence in sentences:
             candidate_sentences[sentence] = sentence.lower()
         candidate_sentences
```

```
Out[93]: {'"I have a degree from Columbia, and now I have to get one from America."': '"i hav
         i have to get one from america."',
          '- Jeff Winger, Community.': '- jeff winger, community.',
          '- The truffle egg dish: had sausage, potatoes, some leafy greens, sautéed mushroo
         he truffle egg dish: had sausage, potatoes, some leafy greens, sautéed mushrooms, a
          '- Troy Barnes Man I love that show.': '- troy barnes man i love that show.',
          '-7 grains waffle: had vanilla cream, raspberry, cherry, and toasted nuts.': '-7 g
         aspberry, cherry, and toasted nuts.',
          '15 minuets wait for the wait staff, 45 minuets before we got the wrong dish!': '1
         45 minuets before we got the wrong dish!',
          '2 HOURS AND NO FOOD!': '2 hours and no food!',
```

```
In [94]: most_frequent_words
```

```
Out[94]: [('i', 73),
          ('the', 46),
          ('food', 20),
          ('brunch', 20),
          ('good', 19),
          ('place', 15),
```

```
In [96]: for long, short in candidate_sentences.items():
             count = 0
             for freq_word, frequency_score in most_frequent_words:
                 if freq_word in short:
                     count += frequency_score
             candidate_sentence_counts[long] = count
```

```
In [97]: candidate_sentence_counts
```

```
Out[97]: {'"I have a degree from Columbia, and now I have to get one
          '- Jeff Winger, Community.': 83,
          '- The truffle egg dish: had sausage, potatoes, some leafy
          '- Troy Barnes Man I love that show.': 73,
          '-7 grains waffle: had vanilla cream, raspberry, cherry, a
```

Text Summarization

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

In [98]:
```
sorted_sentences = OrderedDict(sorted(
                    candidate_sentence_counts.items(),
                    key = lambda x: x[1],
                    reverse = True)[:4])
pp.pprint(sorted_sentences)
```

```
OrderedDict([    (    "I've come here several times with a friend for brunch and "
                      "once for dinner -- we've both really enjoyed a lot of the "
                      'breakfast foods available; my favorite is the brioche '
                      'French toast with blackberry and lemon curd & she loves '
                      'heir waffles.',
```

Packaging all this into a function

In [ ]:
```
def build_naive_summary(text):
    from nltk.tokenize import word_tokenize
    from nltk.tokenize import sent_tokenize
    from nltk.probability import FreqDist
    from nltk.corpus import stopwords
    from collections import OrderedDict
    summary_sentences = []
    candidate_sentences = {}
    candidate_sentence_counts = {}
    striptext = text.replace('\n\n', ' ')
    striptext = striptext.replace('\n', ' ')
    words = word_tokenize(striptext)
    lowercase_words = [word.lower() for word in words
                       if word not in stopwords.words() and word.isalpha()]
    word_frequencies = FreqDist(lowercase_words)
    most_frequent_words = FreqDist(lowercase_words).most_common(20)
    sentences = sent_tokenize(striptext)
    for sentence in sentences:
        candidate_sentences[sentence] = sentence.lower()
    for long, short in candidate_sentences.items():
        count = 0
        for freq_word, frequency_score in most_frequent_words:
            if freq_word in short:
                count += frequency_score
                candidate_sentence_counts[long] = count
    sorted_sentences = OrderedDict(sorted(
                        candidate_sentence_counts.items(),
                        key = lambda x: x[1],
                        reverse = True)[:4])
    return sorted_sentences
```

In [101]:
```
summary = '\n'.join(build_naive_summary(community_data.raw()))
print(summary)
```

```
I've come here several times with a friend for brunch and once for dinner -- we
reakfast foods available; my favorite is the brioche French toast with blackber
fles.
There are a lot of nice places in the city where you can get a very good breakf
ty.
Came here for brunch with my wife after she found the good review, and it did r
The beans were seasoned perfectly, as was the rest of the tomato sauce, The tor
with delicious guacamole and sour cream.
```

In [102]:
```
summary = '\n'.join(build_naive_summary(le_monde_data.raw()))
print(summary)
```

```
i would recommend going for brunch over any other meal, as everything i've orde
shakshuka is great, and you can't anything else like it in the area.
I would love to try them out for dinner, as I said, the service was great and v
I would give them a better review because the food was good but the waiting rea
The food was really good and they are known for their brunch menu.
```

Summarizing President's speech

In [104]:
```
build_naive_summary(inaugural.raw('2013-Obama.txt'))
```

Gensim - Text Summarizer

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

In [107]:
```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
%matplotlib inline
import nltk
from nltk.corpus import PlaintextCorpusReader
from nltk import sent_tokenize,word_tokenize
from nltk.book import *
```

In [108]:
```
import nltk
from nltk.corpus import PlaintextCorpusReader
community_root = "data/community"
le_monde_root = "data/le_monde"
community_files = "community.*"
le_monde_files = "le_monde.*"
heights_root = "data/heights"
heights_files = "heights.*"
amigos_root = "data/amigos"
amigos_files = "amigos.*"
community_data = PlaintextCorpusReader(community_root,community_files)
le_monde_data = PlaintextCorpusReader(le_monde_root,le_monde_files)
heights_data = PlaintextCorpusReader(heights_root,heights_files)
amigos_data = PlaintextCorpusReader(amigos_root,amigos_files)
```

In [109]:
```
type(community_data)
```

Out[109]:
```
nltk.corpus.reader.plaintext.PlaintextCorpusReader
```

In [ ]:
```
text = community_data.raw()
summary_sentences = []
candidate_sentences = {}
candidate_sentence_counts = {}
striptext = text.replace('\n\n', ' ')
striptext = striptext.replace('\n', ' ')
```

In [*]:
```
!pip install gensim
```

In [ ]:
```
import gensim.summarization
```

In [113]:
```
summary = gensim.summarization.summarize(striptext, word_count=100)
print(summary)
```

```
I've come here several times with a friend for brunch and once for dinner -- we've both really enjoyed a lot of the b
reakfast foods available; my favorite is the brioche French toast with blackberry and lemon curd & she loves heir waf
fles.
I ordered the Country Breakfast and the eggs were delicious and fluffy, the biscuit was moist and flavorful, the carr
ot hash browns were warm and comforting, and the ham was sweet and juicy.
This place is really good for breakfast and has great pancakes, sausages, eggs.
```

Topic Modeling

The goal of topic modeling is to identify the major concepts underlying a piece of text
Topic modelling uses Unsupervised Learning. No prior knowledge is necessay, though it is helpful in cleaning up results!

LDA: Latent Dirichlet Allocation Model

- Identifies potential topics using pruning techniques like "upward closure"
- Computes conditional probablities for topic word sets
- Identifies the most likely topics
- Does this over multiple passes probabilities for topic
- Good intuitive explanation: http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/

In [116]:
```
from gensim import corpora
from gensim.models.ldamodel import LdaModel
from gensim.parsing.preprocessing import STOPWORDS
import pprint
```

Topic Modeling

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Prepare the text

In [117]:
```
text = PlaintextCorpusReader("data/","Nikon_coolpix_4300.txt").raw()
striptext = text.replace('\n\n', ' ')
striptext = striptext.replace('\n', ' ')
sentences = sent_tokenize(striptext)
#words = word_tokenize(striptext)
#tokenize each sentence into word tokens
texts = [[word for word in sentence.lower().split()
        if word not in STOPWORDS and word.isalnum()]
        for sentence in sentences]
len(texts)
```

Out[117]: 360

In [118]:
```
texts
```

Out[118]:
```
[['annotated', 'minqing', 'hu', 'bing'],
 ['department',
  'sicence',
  'university',
  'illinois',
  'chicago',
  'product',
  'nikon',
  'coolpix',
```

Create a (word,frequency) dictionary for each word in the text

In [ ]:
```
print(text)
```

In [ ]:
```
text
```

In [126]:
```
dictionary = corpora.Dictionary(texts) #(word_id,frequency) pairs
corpus = [dictionary.doc2bow(text) for text in texts] #(word_id,freq) pairs by sentence
#print(dictionary.token2id)
#print(dictionary.keys())
print(corpus[9])
#print(texts[9])
#print(dictionary[73])
#dictionary[4]
```

```
[(17, 1), (27, 1), (66, 2), (67, 1), (68, 1), (69, 1), (70, 1), (71, 1), (72, 1), (73, 1), (74, 1)]
```

In [122]:

Out[122]: <gensim.corpora.dictionary.Dictionary at 0x122401e80>

Do the LDA

Parameters:

- Number of topics: The number of topics we have generated. The larger the document, the more desirable topics
- Passes: The LDA model makes through the document. More passes, slower analysis

```
In [132]: #Set parameters
          num_topics = 5 #The number of topics that should be generated
          passes = 10
```

```
In [133]: lda = LdaModel(corpus,
                         id2word=dictionary,
                         num_topics=num_topics,
                         passes=10)
```

```
In [ ]:
```

See results

```
In [134]: pp = pprint.PrettyPrinter(indent=4)
          pp.pprint(lda.print_topics(num_words=3))
```

```
[    (0, '0.039*"camera" + 0.020*"pictures" + 0.019*"card"'),
     (1, '0.035*"camera" + 0.025*"nikon" + 0.017*"4300"'),
     (2, '0.021*"pics" + 0.010*"use" + 0.010*"camera"'),
     (3, '0.049*"camera" + 0.022*"battery" + 0.012*"little"'),
     (4, '0.038*"camera" + 0.019*"picture" + 0.016*"use"')]
```

See results

```
In [135]: pp = pprint.PrettyPrinter(indent=4)
          pp.pprint(lda.print_topics(num_words=5))
```

```
[    (   0,
         '0.039*"camera" + 0.020*"pictures" + 0.019*"card" + 0.011*"manual" + '
         '0.009*"digital"'),
     (   1,
         '0.035*"camera" + 0.025*"nikon" + 0.017*"4300" + 0.016*"digital" + '
         '0.013*"coolpix"'),
     (   2,
         '0.021*"pics" + 0.010*"use" + 0.010*"camera" + 0.009*"quality" + '
         '0.008*"settings"'),
     (   3,
         '0.049*"camera" + 0.022*"battery" + 0.012*"little" + 0.012*"use" + '
         '0.012*"great"'),
     (   4,
         '0.038*"camera" + 0.019*"picture" + 0.016*"use" + 0.014*"nikon" + '
```

Do the LDA

We're using sentences as documents here, so this is less than ideal

```
In [140]: from operator import itemgetter
          lda.get_document_topics(corpus[0],minimum_probability=0.05,per_word_topics=False)
          sorted(lda.get_document_topics(corpus[3],minimum_probability=0,per_word_topics=False),key=itemgetter(1),reverse=True)

Out[140]: [(0, 0.72795902889073394),
           (3, 0.069169371008370825),
           (1, 0.068004387432491806),
           (2, 0.067912472945450372),
           (4, 0.066954739722953058)]
```

Making Sense of the Topic

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Draw wordclouds

In [ ]:
```
def draw_wordcloud(lda,topicnum,min_size=0,STOPWORDS=[]):
    word_list=[]
    prob_total = 0
    for word,prob in lda.show_topic(topicnum,topn=50):
        prob_total +=prob
    for word,prob in lda.show_topic(topicnum,topn=50):
        if word in STOPWORDS or  len(word) < min_size:
            continue
        freq = int(prob/prob_total*1000)
        alist=[word]
        word_list.extend(alist*freq)

    from wordcloud import WordCloud, STOPWORDS
    import matplotlib.pyplot as plt
    %matplotlib inline
    text = ' '.join(word_list)
    wordcloud = WordCloud(stopwords=STOPWORDS,background_color='white',width=3000,height=3000).generate('


    plt.imshow(wordcloud)
    plt.axis('off')
    plt.show()
```

In [143]:
```
draw_wordcloud(lda,2)
```

Roughly,

- lda looks for candidate topics assuming that there are many such candidates
- looks for words related to the candidate topics
- assign probablilites to those words

Presidential Addresses

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Let's look at Presidential addresses to see what sorts of topics emerge from there

- Each document will be analyzed for topic
- The corpus will consist of 58 documents, one per presidential address

In [ ]:
```
REMOVE_WORDS = {'shall','generally','spirit','country','people','nation','nations','great','better'}
#Create a word dictionary (id, word)
texts = [[word for word in sentence.lower().split()
        if word not in STOPWORDS and word not in REMOVE_WORDS and word.isalnum()]
        for sentence in sentences]
dictionary = corpora.Dictionary(texts)

#Create a corpus of documents
text_list = list()
for fileid in inaugural.fileids():
    text = inaugural.words(fileid)
    doc=list()
    for word in text:
        if word in STOPWORDS or word in REMOVE_WORDS or not word.isalpha() or len(word) <5:
            continue
        doc.append(word)
    text_list.append(doc)
by_address_corpus = [dictionary.doc2bow(text) for text in text_list]
```

In [146]:
```
len(by_address_corpus)
```

Out[146]: 58

Presidential Addresses: Create the Model

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

In [147]:
```
lda = LdaModel(by_address_corpus,
               id2word=dictionary,
               num_topics=20,
               passes=10)
```

In [148]:
```
pp = pprint.PrettyPrinter(indent=4)
pp.pprint(lda.print_topics(num_words=10))
```

```
[   (   0,
        '0.032*"course" + 0.030*"things" + 0.028*"knowledge" + 0.023*"process" '
        '+ 0.020*"years" + 0.019*"order" + 0.019*"little" + 0.017*"thought" + '
        '0.017*"support" + 0.017*"familiar"'),
    (   1,
        '0.003*"world" + 0.002*"power" + 0.002*"place" + 0.001*"future" + '
        '0.001*"children" + 0.001*"course" + 0.001*"years" + 0.001*"service" + '
        '0.001*"office" + 0.001*"order"'),
    (   2,
        '0.105*"power" + 0.026*"control" + 0.024*"given" + 0.019*"hands" + '
        '0.016*"state" + 0.014*"subject" + 0.013*"produce" + 0.013*"important" '
        '+ 0.013*"department" + 0.013*"intended"'),
    (   3,
        '0.043*"years" + 0.031*"power" + 0.028*"action" + 0.019*"future" + '
        '0.018*"office" + 0.017*"service" + 0.017*"period" + 0.016*"opinion" + '
        '0.015*"trust" + 0.015*"ability"'),
    (   4,
        '0.033*"office" + 0.022*"state" + 0.022*"highest" + 0.022*"recommend" '
        '+ 0.022*"performance" + 0.012*"believe" + 0.011*"right" + '
        '0.011*"power" + 0.011*"control" + 0.011*"action"'),
    (   5,
        '0.035*"world" + 0.028*"support" + 0.022*"service" + 0.021*"power" + '
        '0.017*"years" + 0.017*"given" + 0.017*"important" + 0.016*"order" + '
```

Presidential Addresses: Create the Model

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

```
In [ ]: len(by_address_corpus)
```

```
In [152]: from operator import itemgetter
          sorted(lda.get_document_topics(by_address_corpus[56],minimum_probability=0,per_word_topics=False),key=itemgetter(1),rev
```

```
Out[152]: [(8, 0.51051380624923337),
           (11, 0.48152159191111937),
           (5, 0.00044247788373801006),
           (17, 0.00044247788355170585),
           (19, 0.00044247788285562616),
           (12, 0.00044247788249148379),
           (13, 0.00044247788194282128),
           (3, 0.00044247788192305253),
           (0, 0.00044247788185107641),
           (2, 0.00044247788815170437),
           (9, 0.00044247788129546736),
           (6, 0.00044247788065216576),
           (10, 0.00044247787985220237),
           (15, 0.00044247787900034022),
           (4, 0.00044247787791450085),
           (14, 0.00044247787663700809),
           (1, 0.00044247787610634767),
           (7, 0.00044247787610627199),
           (16, 0.00044247787610620217),
           (18, 0.00044247787610619501)]
```

```
In [153]: draw_wordcloud(lda,11)
```


```
In [ ]: print(lda.show_topic(12,topn=5))
        print(lda.show_topic(18,topn=5))
```

Presidential Addresses: Create the Model

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

Given a corpus of documents, when a new document arrives, find the document that is the most similar

In [ ]:
```
doc_list = [community_data,le_monde_data,amigos_data,heights_data]
all_text = community_data.raw() + le_monde_data.raw() + amigos_data.raw() + heights_data.raw()

documents = [doc.raw() for doc in doc_list]
texts = [[word for word in document.lower().split()
        if word not in STOPWORDS and word.isalnum()]
        for document in documents]
dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]
```

In [ ]:
```
from gensim.similarities.docsim import Similarity
from gensim import corpora, models, similarities
lsi = models.LsiModel(corpus, id2word=dictionary, num_topics=2)
doc = """
Many, many years ago, I used to frequent this place for their amazing french toast.
It's been a while since then and I've been hesitant to review a place I haven't been to in 7-8 years...
but I passed by French Roast and, feeling nostalgic, decided to go back.

It was a great decision.

Their Bloody Mary is fantastic and includes bacon (which was perfectly cooked!!), olives,
cucumber, and celery. The Irish coffee is also excellent, even without the cream which is what I ordered.

Great food, great drinks, a great ambiance that is casual yet familiar like a tiny little French cafe.
I highly recommend coming here, and will be back whenever I'm in the area next.

Juan, the bartender, is great!! One of the best in any brunch spot in the city, by far.
"""
vec_bow = dictionary.doc2bow(doc.lower().split())
vec_lsi = lsi[vec_bow]
index = similarities.MatrixSimilarity(lsi[corpus])
sims = index[vec_lsi]
sims = sorted(enumerate(sims), key=lambda item: -item[1])
```

Presidential Addresses: Create the Model

COLUMBIA | ENGINEERING
EXECUTIVE EDUCATION

In [156]:
```
sims
```

Out[156]:
```
[(1, 0.98765731), (0, 0.95548683), (3, 0.79290682), (2, 0.7698077)]
```

In [ ]:
```
doc="""
I went to Mexican Festival Restaurant for Cinco De Mayo because I had been there years
prior and had such a good experience. This time wasn't so good. The food was just
mediocre and it wasn't hot when it was brought to our table. They brought my friends food out
10 minutes before everyone else and it took forever to get drinks. We let it slide because the place was
packed with people and it was Cinco De Mayo. Also, the margaritas we had were slamming! Pure tequila.

But then things took a turn for the worst. As I went to get something out of my purse which was on
the back of my chair, I looked down and saw a huge water bug. I had to warn the lady next to me because
it was so close to her chair. We called the waitress over and someone came with a broom and a dustpan and
swept it away like it was an everyday experience. No one seemed phased.

Even though our waitress was very nice, I do not think we will be returning to Mexican Festival again.
It seems the restaurant is a shadow of its former self.
"""
vec_bow = dictionary.doc2bow(doc.lower().split())
vec_lsi = lsi[vec_bow]
index = similarities.MatrixSimilarity(lsi[corpus])
sims = index[vec_lsi]
sims = sorted(enumerate(sims), key=lambda item: -item[1])
```

www.emeritus.org