Project 5: NLP on Financial Statements

Instructions

Each problem consists of a function to implement and instructions on how to implement the function. The parts of the function that need to be implemented are marked with a `# TODO` comment. After implementing the function, run the cell to test it against the unit tests we've provided. For each problem, we provide one or more unit tests from our `project_tests` package. These unit tests won't tell you if your answer is correct, but will warn you of any major errors. Your code will be checked for the correct solution when you submit it to Udacity.

Packages

When you implement the functions, you'll only need to you use the packages you've used in the classroom, like [Pandas (https://pandas.pydata.org/)](https://pandas.pydata.org/) and [Numpy (http://www.numpy.org/)](http://www.numpy.org/). These packages will be imported for you. We recommend you don't add any import statements, otherwise the grader might not be able to run your code.

The other packages that we're importing are `project_helper` and `project_tests` . These are custom packages built to help you solve the problems. The `project_helper` module contains utility functions and graph functions. The `project_tests` contains the unit tests for all the problems.

Install Packages

In [1]:
```
import sys
!{sys.executable} -m pip install -r requirements.txt
```

```
Requirement already satisfied: alphalens==0.3.2 in /opt/conda/lib/python3.6/site-packages (from -r requiremen
ts.txt (line 1)) (0.3.2)
Requirement already satisfied: nltk==3.3.0 in /opt/conda/lib/python3.6/site-packages (from -r requirements.tx
t (line 2)) (3.3)
Requirement already satisfied: numpy==1.13.3 in /opt/conda/lib/python3.6/site-packages (from -r requirements.
txt (line 3)) (1.13.3)
Requirement already satisfied: ratelimit==2.2.0 in /opt/conda/lib/python3.6/site-packages (from -r requiremen
ts.txt (line 4)) (2.2.0)
Requirement already satisfied: requests==2.18.4 in /opt/conda/lib/python3.6/site-packages (from -r requiremen
ts.txt (line 5)) (2.18.4)
Requirement already satisfied: scikit-learn==0.19.1 in /opt/conda/lib/python3.6/site-packages (from -r requir
ements.txt (line 6)) (0.19.1)
Requirement already satisfied: six==1.11.0 in /opt/conda/lib/python3.6/site-packages (from -r requirements.tx
t (line 7)) (1.11.0)
Requirement already satisfied: tqdm==4.19.5 in /opt/conda/lib/python3.6/site-packages (from -r requirements.t
xt (line 8)) (4.19.5)
Requirement already satisfied: matplotlib>=1.4.0 in /opt/conda/lib/python3.6/site-packages (from alphalens==
0.3.2->-r requirements.txt (line 1)) (2.1.0)
Requirement already satisfied: statsmodels>=0.6.1 in /opt/conda/lib/python3.6/site-packages (from alphalens==
0.3.2->-r requirements.txt (line 1)) (0.8.0)
Requirement already satisfied: seaborn>=0.6.0 in /opt/conda/lib/python3.6/site-packages (from alphalens==0.3.
2->-r requirements.txt (line 1)) (0.8.1)
Requirement already satisfied: pandas>=0.18.0 in /opt/conda/lib/python3.6/site-packages (from alphalens==0.3.
2->-r requirements.txt (line 1)) (0.23.3)
Requirement already satisfied: IPython>=3.2.3 in /opt/conda/lib/python3.6/site-packages (from alphalens==0.3.
2->-r requirements.txt (line 1)) (6.5.0)
Requirement already satisfied: scipy>=0.14.0 in /opt/conda/lib/python3.6/site-packages (from alphalens==0.3.2
->-r requirements.txt (line 1)) (0.19.1)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/conda/lib/python3.6/site-packages (from requests
==2.18.4->-r requirements.txt (line 5)) (3.0.4)
Requirement already satisfied: idna<2.7,>=2.5 in /opt/conda/lib/python3.6/site-packages (from requests==2.18.
4->-r requirements.txt (line 5)) (2.6)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /opt/conda/lib/python3.6/site-packages (from requests
==2.18.4->-r requirements.txt (line 5)) (1.22)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.6/site-packages (from requests==
2.18.4->-r requirements.txt (line 5)) (2019.6.16)
Requirement already satisfied: python-dateutil>=2.0 in /opt/conda/lib/python3.6/site-packages (from matplotli
b>=1.4.0->alphalens==0.3.2->-r requirements.txt (line 1)) (2.6.1)
Requirement already satisfied: pytz in /opt/conda/lib/python3.6/site-packages (from matplotlib>=1.4.0->alphal
ens==0.3.2->-r requirements.txt (line 1)) (2017.3)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.6/site-packages/cycler-0.10.0-py3.6.egg
(from matplotlib>=1.4.0->alphalens==0.3.2->-r requirements.txt (line 1)) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /opt/conda/lib/python3.6/site-pack
```

```
ages (from matplotlib>=1.4.0->alphalens==0.3.2->-r requirements.txt (line 1)) (2.2.0)
Requirement already satisfied: decorator in /opt/conda/lib/python3.6/site-packages (from IPython>=3.2.3->alph
alens==0.3.2->-r requirements.txt (line 1)) (4.0.11)
Requirement already satisfied: pexpect; sys_platform != "win32" in /opt/conda/lib/python3.6/site-packages (fr
om IPython>=3.2.3->alphalens==0.3.2->-r requirements.txt (line 1)) (4.3.1)
Requirement already satisfied: pickleshare in /opt/conda/lib/python3.6/site-packages (from IPython>=3.2.3->al
phalens==0.3.2->-r requirements.txt (line 1)) (0.7.4)
Requirement already satisfied: jedi>=0.10 in /opt/conda/lib/python3.6/site-packages (from IPython>=3.2.3->alp
halens==0.3.2->-r requirements.txt (line 1)) (0.10.2)
Requirement already satisfied: backcall in /opt/conda/lib/python3.6/site-packages (from IPython>=3.2.3->alpha
lens==0.3.2->-r requirements.txt (line 1)) (0.1.0)
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.15 in /opt/conda/lib/python3.6/site-packages (from
IPython>=3.2.3->alphalens==0.3.2->-r requirements.txt (line 1)) (1.0.15)
Requirement already satisfied: setuptools>=18.5 in /opt/conda/lib/python3.6/site-packages (from IPython>=3.2.
3->alphalens==0.3.2->-r requirements.txt (line 1)) (38.4.0)
Requirement already satisfied: traitlets>=4.2 in /opt/conda/lib/python3.6/site-packages (from IPython>=3.2.3-
>alphalens==0.3.2->-r requirements.txt (line 1)) (4.3.2)
Requirement already satisfied: pygments in /opt/conda/lib/python3.6/site-packages (from IPython>=3.2.3->alpha
lens==0.3.2->-r requirements.txt (line 1)) (2.2.0)
Requirement already satisfied: simplegeneric>0.8 in /opt/conda/lib/python3.6/site-packages (from IPython>=3.
2.3->alphalens==0.3.2->-r requirements.txt (line 1)) (0.8.1)
Requirement already satisfied: ptyprocess>=0.5 in /opt/conda/lib/python3.6/site-packages (from pexpect; sys_p
latform != "win32"->IPython>=3.2.3->alphalens==0.3.2->-r requirements.txt (line 1)) (0.5.2)
Requirement already satisfied: wcwidth in /opt/conda/lib/python3.6/site-packages (from prompt-toolkit<2.0.0,>
=1.0.15->IPython>=3.2.3->alphalens==0.3.2->-r requirements.txt (line 1)) (0.1.7)
Requirement already satisfied: ipython-genutils in /opt/conda/lib/python3.6/site-packages (from traitlets>=4.
2->IPython>=3.2.3->alphalens==0.3.2->-r requirements.txt (line 1)) (0.2.0)
```

Load Packages

In [2]:

```
import nltk
import numpy as np
import pandas as pd
import pickle
import pprint
import project_helper
import project_tests

from tqdm import tqdm
```

Download NLP Corpora

You'll need two corpora to run this project: the stopwords corpus for removing stopwords and wordnet for lemmatizing.

In [3]:
```
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[3]:
```
True
```

Get 10ks

We'll be running NLP analysis on 10-k documents. To do that, we first need to download the documents. For this project, we'll download 10-ks for a few companies. To lookup documents for these companies, we'll use their CIK. If you would like to run this against other stocks, we've provided the dict `additional_cik` for more stocks. However, the more stocks you try, the long it will take to run.

In [4]:
```
cik_lookup = {
    'AMZN': '0001018724',
    'BMY': '0000014272',
    'CNP': '0001130310',
    'CVX': '0000093410',
    'FL': '0000850209',
    'FRT': '0000034903',
    'HON': '0000773840'}

additional_cik = {
    'AEP': '0000004904',
    'AXP': '0000004962',
    'BA': '0000012927',
    'BK': '0001390777',
    'CAT': '0000018230',
    'DE': '0000315189',
    'DIS': '0001001039',
    'DTE': '0000936340',
    'ED': '0001047862',
    'EMR': '0000032604',
    'ETN': '0001551182',
    'GE': '0000040545',
    'IBM': '0000051143',
    'IP': '0000051434',
    'JNJ': '0000200406',
    'KO': '0000021344',
    'LLY': '0000059478',
    'MCD': '0000063908',
    'MO': '0000764180',
    'MRK': '0000310158',
    'MRO': '0000101778',
    'PCG': '0001004980',
    'PEP': '0000077476',
    'PFE': '0000078003',
    'PG': '0000080424',
    'PNR': '0000077360',
    'SYY': '0000096021',
    'TXN': '0000097476',
    'UTX': '0000101829',
    'WFC': '0000072971',
    'WMT': '0000104169',
```

```
    'WY': '0000106535',
    'XOM': '0000034088'}
```

Get list of 10-ks

The SEC has a limit on the number of calls you can make to the website per second. In order to avoid hiding that limit, we've created the `SecAPI` class. This will cache data from the SEC and prevent you from going over the limit.

In [5]:
```
sec_api = project_helper.SecAPI()
```

With the class constructed, let's pull a list of filled 10-ks from the SEC for each company.

In [6]:
```
from bs4 import BeautifulSoup

def get_sec_data(cik, doc_type, start=0, count=60):
    newest_pricing_data = pd.to_datetime('2018-01-01')
    rss_url = 'https://www.sec.gov/cgi-bin/browse-edgar?action=getcompany' \
        '&CIK={}&type={}&start={}&count={}&owner=exclude&output=atom' \
        .format(cik, doc_type, start, count)
    sec_data = sec_api.get(rss_url)
    feed = BeautifulSoup(sec_data.encode('ascii'), 'xml').feed
    entries = [
        (
            entry.content.find('filing-href').getText(),
            entry.content.find('filing-type').getText(),
            entry.content.find('filing-date').getText())
        for entry in feed.find_all('entry', recursive=False)
        if pd.to_datetime(entry.content.find('filing-date').getText()) <= newest_pricing_data]

    return entries
```

Let's pull the list using the `get_sec_data` function, then display some of the results. For displaying some of the data, we'll use Amazon as an example.

In [7]:
```python
example_ticker = 'AMZN'
sec_data = {}

for ticker, cik in cik_lookup.items():
    sec_data[ticker] = get_sec_data(cik, '10-K')

pprint.pprint(sec_data[example_ticker][:5])
```

```
[('https://www.sec.gov/Archives/edgar/data/1018724/000101872417000011/0001018724-17-000011-index.htm',
  '10-K',
  '2017-02-10'),
 ('https://www.sec.gov/Archives/edgar/data/1018724/000101872416000172/0001018724-16-000172-index.htm',
  '10-K',
  '2016-01-29'),
 ('https://www.sec.gov/Archives/edgar/data/1018724/000101872415000006/0001018724-15-000006-index.htm',
  '10-K',
  '2015-01-30'),
 ('https://www.sec.gov/Archives/edgar/data/1018724/000101872414000006/0001018724-14-000006-index.htm',
  '10-K',
  '2014-01-31'),
 ('https://www.sec.gov/Archives/edgar/data/1018724/000119312513028520/0001193125-13-028520-index.htm',
  '10-K',
  '2013-01-30')]
```

Download 10-ks

As you see, this is a list of urls. These urls point to a file that contains metadata related to each filling. Since we don't care about the metadata, we'll pull the filling by replacing the url with the filling url.

In [8]:
```
raw_fillings_by_ticker = {}

for ticker, data in sec_data.items():
    raw_fillings_by_ticker[ticker] = {}
    for index_url, file_type, file_date in tqdm(data, desc='Downloading {} Fillings'.format(ticker), unit='fi
lling'):
        if (file_type == '10-K'):
            file_url = index_url.replace('-index.htm', '.txt').replace('.txtl', '.txt')

            raw_fillings_by_ticker[ticker][file_date] = sec_api.get(file_url)


print('Example Document:\n\n{}...'.format(next(iter(raw_fillings_by_ticker[example_ticker].values()))[:1000
]))
```

```
Downloading AMZN Fillings: 100%|██████████| 22/22 [00:05<00:00,  4.14filling/s]
Downloading BMY Fillings: 100%|██████████| 27/27 [00:08<00:00,  3.11filling/s]
Downloading CNP Fillings: 100%|██████████| 19/19 [00:07<00:00,  2.68filling/s]
Downloading CVX Fillings: 100%|██████████| 25/25 [00:07<00:00,  3.31filling/s]
Downloading FL Fillings: 100%|██████████| 22/22 [00:04<00:00,  4.89filling/s]
Downloading FRT Fillings: 100%|██████████| 29/29 [00:05<00:00,  5.10filling/s]
Downloading HON Fillings: 100%|██████████| 25/25 [00:06<00:00,  4.09filling/s]
```

Example Document:

```
<SEC-DOCUMENT>0001018724-17-000011.txt : 20170210
<SEC-HEADER>0001018724-17-000011.hdr.sgml : 20170210
<ACCEPTANCE-DATETIME>20170209175636
ACCESSION NUMBER:               0001018724-17-000011
CONFORMED SUBMISSION TYPE:      10-K
PUBLIC DOCUMENT COUNT:          92
CONFORMED PERIOD OF REPORT:     20161231
FILED AS OF DATE:               20170210
DATE AS OF CHANGE:              20170209

FILER:

        COMPANY DATA:
                COMPANY CONFORMED NAME:                 AMAZON COM INC
                CENTRAL INDEX KEY:                      0001018724
                STANDARD INDUSTRIAL CLASSIFICATION:     RETAIL-CATALOG & MAIL-ORDER HOUSES [5961]
                IRS NUMBER:                             911646860
                STATE OF INCORPORATION:                 DE
                FISCAL YEAR END:                        1231

        FILING VALUES:
                FORM TYPE:              10-K
                SEC ACT:                1934 Act
                SEC FILE NUMBER:        000-22513
                FILM NUMBER:            17588807

        BUSINESS ADDRESS:
                STREET 1:               410 TERRY AVENUE NORTH
                CITY:                   SEATTLE
                STATE:                  WA
                ZIP:                    98109
                BUSINESS PHONE:         2062661000

        MAIL ADDRESS:
                STREET 1:               410 TERRY AVENUE NORTH
                CITY:                   SEATTLE
                STATE:                  WA
                ZIP:                    98109
</SEC-HEADER>
<DOCUMENT>
<TYPE>10-K
```

```
<SEQUENCE>1
<FILENAME...
```

Get Documents

With theses fillings downloaded, we want to break them into their associated documents. These documents are sectioned off in the fillings with the tags `<DOCUMENT>` for the start of each document and `</DOCUMENT>` for the end of each document. There's no overlap with these documents, so each `</DOCUMENT>` tag should come after the `<DOCUMENT>` with no `<DOCUMENT>` tag in between.

Implement `get_documents` to return a list of these documents from a filling. Make sure not to include the tag in the returned document text.

In [9]:
```
import re


def get_documents(text):
    """
    Extract the documents from the text

    Parameters
    ----------
    text : str
        The text with the document strings inside

    Returns
    -------
    extracted_docs : list of str
        The document strings found in `text`
    """

    # TODO: Implement
    regex_start_doc = re.compile(r'<DOCUMENT>')
    regex_end_doc = re.compile(r'</DOCUMENT>')

    extracted_docs_start_idx = [x.end() for x in regex_start_doc.finditer(text)]
    extracted_docs_end_idx   = [x.start() for x in regex_end_doc.finditer(text)]

    return [text[extracted_docs_start_idx[i] : extracted_docs_end_idx[i]] for i in range(len(extracted_docs_s
tart_idx))]

project_tests.test_get_documents(get_documents)
```

```
Tests Passed
```

With the `get_documents` function implemented, let's extract all the documents.

In [10]:

```
filling_documents_by_ticker = {}

for ticker, raw_fillings in raw_fillings_by_ticker.items():
    filling_documents_by_ticker[ticker] = {}
    for file_date, filling in tqdm(raw_fillings.items(), desc='Getting Documents from {} Fillings'.format(tic
ker), unit='filling'):
        filling_documents_by_ticker[ticker][file_date] = get_documents(filling)


print('\n\n'.join([
    'Document {} Filed on {}:\n{}...'.format(doc_i, file_date, doc[:200])
    for file_date, docs in filling_documents_by_ticker[example_ticker].items()
    for doc_i, doc in enumerate(docs)][:3]))
```

```
Getting Documents from AMZN Fillings: 100%|██████████| 17/17 [00:00<00:00, 85.84filling/s]
Getting Documents from BMY Fillings: 100%|██████████| 23/23 [00:00<00:00, 40.55filling/s]
Getting Documents from CNP Fillings: 100%|██████████| 15/15 [00:00<00:00, 37.36filling/s]
Getting Documents from CVX Fillings: 100%|██████████| 21/21 [00:00<00:00, 37.44filling/s]
Getting Documents from FL Fillings: 100%|██████████| 16/16 [00:00<00:00, 50.78filling/s]
Getting Documents from FRT Fillings: 100%|██████████| 19/19 [00:00<00:00, 25.96filling/s]
Getting Documents from HON Fillings: 100%|██████████| 20/20 [00:00<00:00, 20.50filling/s]

Document 0 Filed on 2017-02-10:

<TYPE>10-K
<SEQUENCE>1
<FILENAME>amzn-20161231x10k.htm
<DESCRIPTION>FORM 10-K
<TEXT>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
        <he...

Document 1 Filed on 2017-02-10:

<TYPE>EX-12.1
<SEQUENCE>2
<FILENAME>amzn-20161231xex121.htm
<DESCRIPTION>COMPUTATION OF RATIO OF EARNINGS TO FIXED CHARGES
<TEXT>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http:...

Document 2 Filed on 2017-02-10:

<TYPE>EX-21.1
<SEQUENCE>3
<FILENAME>amzn-20161231xex211.htm
<DESCRIPTION>LIST OF SIGNIFICANT SUBSIDIARIES
<TEXT>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/h...
```

Get Document Types

Now that we have all the documents, we want to find the 10-k form in this 10-k filing. Implement the `get_document_type` function to return the type of document given. The document type is located on a line with the `<TYPE>` tag. For example, a form of type "TEST" would have the line `<TYPE>TEST`. Make sure to return the type as lowercase, so this example would be returned as "test".

In [11]:
```
def get_document_type(doc):
    """
    Return the document type lowercased

    Parameters
    ----------
    doc : str
        The document string

    Returns
    -------
    doc_type : str
        The document type lowercased
    """

    # TODO: Implement
    regex = re.compile(r'<TYPE>[^\n]+')
    type = [x[len('<TYPE>'):].lower() for x in regex.findall(doc)]
    return type[0] if type else ''


project_tests.test_get_document_type(get_document_type)
```

```
Tests Passed
```

With the `get_document_type` function, we'll filter out all non 10-k documents.

In [12]:
```
ten_ks_by_ticker = {}

for ticker, filling_documents in filling_documents_by_ticker.items():
    ten_ks_by_ticker[ticker] = []
    for file_date, documents in filling_documents.items():
        for document in documents:
            if get_document_type(document) == '10-k':
                ten_ks_by_ticker[ticker].append({
                    'cik': cik_lookup[ticker],
                    'file': document,
                    'file_date': file_date})


project_helper.print_ten_k_data(ten_ks_by_ticker[example_ticker][:5], ['cik', 'file', 'file_date'])
```

```
[
  {
    cik: '0001018724'
    file: '\n<TYPE>10-K\n<SEQUENCE>1\n<FILENAME>amzn-2016123...
    file_date: '2017-02-10'},
  {
    cik: '0001018724'
    file: '\n<TYPE>10-K\n<SEQUENCE>1\n<FILENAME>amzn-2015123...
    file_date: '2016-01-29'},
  {
    cik: '0001018724'
    file: '\n<TYPE>10-K\n<SEQUENCE>1\n<FILENAME>amzn-2014123...
    file_date: '2015-01-30'},
  {
    cik: '0001018724'
    file: '\n<TYPE>10-K\n<SEQUENCE>1\n<FILENAME>amzn-2013123...
    file_date: '2014-01-31'},
  {
    cik: '0001018724'
    file: '\n<TYPE>10-K\n<SEQUENCE>1\n<FILENAME>d445434d10k....
    file_date: '2013-01-30'},
]
```

Preprocess the Data

Clean Up

As you can see, the text for the documents are very messy. To clean this up, we'll remove the html and lowercase all the text.

In [13]:
```
def remove_html_tags(text):
    text = BeautifulSoup(text, 'html.parser').get_text()

    return text


def clean_text(text):
    text = text.lower()
    text = remove_html_tags(text)

    return text
```

Using the `clean_text` function, we'll clean up all the documents.

In [14]:
```python
for ticker, ten_ks in ten_ks_by_ticker.items():
    for ten_k in tqdm(ten_ks, desc='Cleaning {} 10-Ks'.format(ticker), unit='10-K'):
        ten_k['file_clean'] = clean_text(ten_k['file'])


project_helper.print_ten_k_data(ten_ks_by_ticker[example_ticker][:5], ['file_clean'])
```

```
Cleaning AMZN 10-Ks: 100%|██████████| 17/17 [00:35<00:00,  2.12s/10-K]
Cleaning BMY 10-Ks: 100%|██████████| 23/23 [01:15<00:00,  3.30s/10-K]
Cleaning CNP 10-Ks: 100%|██████████| 15/15 [00:56<00:00,  3.80s/10-K]
Cleaning CVX 10-Ks: 100%|██████████| 21/21 [01:54<00:00,  5.45s/10-K]
Cleaning FL 10-Ks: 100%|██████████| 16/16 [00:26<00:00,  1.67s/10-K]
Cleaning FRT 10-Ks: 100%|██████████| 19/19 [00:55<00:00,  2.90s/10-K]
Cleaning HON 10-Ks: 100%|██████████| 20/20 [01:01<00:00,  3.07s/10-K]

[
  {
    file_clean: '\n10-k\n1\namzn-20161231x10k.htm\nform 10-k\n\n\n...},
  {
    file_clean: '\n10-k\n1\namzn-20151231x10k.htm\nform 10-k\n\n\n...},
  {
    file_clean: '\n10-k\n1\namzn-20141231x10k.htm\nform 10-k\n\n\n...},
  {
    file_clean: '\n10-k\n1\namzn-20131231x10k.htm\nform 10-k\n\n\n...},
  {
    file_clean: '\n10-k\n1\nd445434d10k.htm\nform 10-k\n\n\nform 1...},
]
```

Lemmatize

With the text cleaned up, it's time to distill the verbs down. Implement the `lemmatize_words` function to lemmatize verbs in the list of words provided.

In [15]:

```python
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet


def lemmatize_words(words):
    """
    Lemmatize words

    Parameters
    ----------
    words : list of str
        List of words

    Returns
    -------
    lemmatized_words : list of str
        List of lemmatized words
    """

    # TODO: Implement

    return [WordNetLemmatizer().lemmatize(word, pos = 'v') for word in words]


project_tests.test_lemmatize_words(lemmatize_words)
```

```
Tests Passed
```

With the `lemmatize_words` function implemented, let's lemmatize all the data.

```
In [16]: word_pattern = re.compile('\w+')

         for ticker, ten_ks in ten_ks_by_ticker.items():
             for ten_k in tqdm(ten_ks, desc='Lemmatize {} 10-Ks'.format(ticker), unit='10-K'):
                 ten_k['file_lemma'] = lemmatize_words(word_pattern.findall(ten_k['file_clean']))


         project_helper.print_ten_k_data(ten_ks_by_ticker[example_ticker][:5], ['file_lemma'])
```

```
Lemmatize AMZN 10-Ks: 100%|██████████| 17/17 [00:05<00:00,  3.2910-K/s]
Lemmatize BMY 10-Ks: 100%|██████████| 23/23 [00:11<00:00,  1.9810-K/s]
Lemmatize CNP 10-Ks: 100%|██████████| 15/15 [00:09<00:00,  1.5510-K/s]
Lemmatize CVX 10-Ks: 100%|██████████| 21/21 [00:10<00:00,  1.9610-K/s]
Lemmatize FL 10-Ks: 100%|██████████| 16/16 [00:04<00:00,  3.7110-K/s]
Lemmatize FRT 10-Ks: 100%|██████████| 19/19 [00:06<00:00,  2.7810-K/s]
Lemmatize HON 10-Ks: 100%|██████████| 20/20 [00:06<00:00,  3.0910-K/s]

[
  {
    file_lemma: '['10', 'k', '1', 'amzn', '20161231x10k', 'htm', '...},
  {
    file_lemma: '['10', 'k', '1', 'amzn', '20151231x10k', 'htm', '...},
  {
    file_lemma: '['10', 'k', '1', 'amzn', '20141231x10k', 'htm', '...},
  {
    file_lemma: '['10', 'k', '1', 'amzn', '20131231x10k', 'htm', '...},
  {
    file_lemma: '['10', 'k', '1', 'd445434d10k', 'htm', 'form', '1...},
]
```

Remove Stopwords

In [17]:
```python
from nltk.corpus import stopwords


lemma_english_stopwords = lemmatize_words(stopwords.words('english'))

for ticker, ten_ks in ten_ks_by_ticker.items():
    for ten_k in tqdm(ten_ks, desc='Remove Stop Words for {} 10-Ks'.format(ticker), unit='10-K'):
        ten_k['file_lemma'] = [word for word in ten_k['file_lemma'] if word not in lemma_english_stopwords]


print('Stop Words Removed')
```

```
Remove Stop Words for AMZN 10-Ks: 100%|██████████| 17/17 [00:02<00:00,  7.8510-K/s]
Remove Stop Words for BMY 10-Ks: 100%|██████████| 23/23 [00:04<00:00,  4.7910-K/s]
Remove Stop Words for CNP 10-Ks: 100%|██████████| 15/15 [00:03<00:00,  3.8310-K/s]
Remove Stop Words for CVX 10-Ks: 100%|██████████| 21/21 [00:04<00:00,  4.5710-K/s]
Remove Stop Words for FL 10-Ks: 100%|██████████| 16/16 [00:01<00:00,  8.7310-K/s]
Remove Stop Words for FRT 10-Ks: 100%|██████████| 19/19 [00:02<00:00,  6.3610-K/s]
Remove Stop Words for HON 10-Ks: 100%|██████████| 20/20 [00:02<00:00,  7.0710-K/s]

Stop Words Removed
```

Analysis on 10ks

Loughran McDonald Sentiment Word Lists

We'll be using the Loughran and McDonald sentiment word lists. These word lists cover the following sentiment:

- Negative
- Positive
- Uncertainty
- Litigious
- Constraining
- Superfluous
- Modal

This will allow us to do the sentiment analysis on the 10-ks. Let's first load these word lists. We'll be looking into a few of these sentiments.

In [18]:
```
import os


sentiments = ['negative', 'positive', 'uncertainty', 'litigious', 'constraining', 'interesting']

sentiment_df = pd.read_csv(os.path.join('..', '..', 'data', 'project_5_loughran_mcdonald', 'loughran_mcdonald
_master_dic_2016.csv'))
sentiment_df.columns = [column.lower() for column in sentiment_df.columns] # Lowercase the columns for ease o
f use

# Remove unused information
sentiment_df = sentiment_df[sentiments + ['word']]
sentiment_df[sentiments] = sentiment_df[sentiments].astype(bool)
sentiment_df = sentiment_df[(sentiment_df[sentiments]).any(1)]

# Apply the same preprocessing to these words as the 10-k words
sentiment_df['word'] = lemmatize_words(sentiment_df['word'].str.lower())
sentiment_df = sentiment_df.drop_duplicates('word')


sentiment_df.head()
```

Out[18]:

	negative	positive	uncertainty	litigious	constraining	interesting	word
9	True	False	False	False	False	False	abandon
12	True	False	False	False	False	False	abandonment
13	True	False	False	False	False	False	abandonments
51	True	False	False	False	False	False	abdicate
54	True	False	False	False	False	False	abdication

Bag of Words

using the sentiment word lists, let's generate sentiment bag of words from the 10-k documents. Implement `get_bag_of_words` to generate a bag of words that counts the number of sentiment words in each doc. You can ignore words that are not in `sentiment_words` .

In [19]:

```python
from collections import defaultdict, Counter
from sklearn.feature_extraction.text import CountVectorizer
import nltk

def get_bag_of_words(sentiment_words, docs):
    """
    Generate a bag of words from documents for a certain sentiment

    Parameters
    ----------
    sentiment_words: Pandas Series
        Words that signify a certain sentiment
    docs : list of str
        List of documents used to generate bag of words

    Returns
    -------
    bag_of_words : 2-d Numpy Ndarray of int
        Bag of words sentiment for each document
        The first dimension is the document.
        The second dimension is the word.
    """

    # TODO: Implement
    vectorizer = CountVectorizer(vocabulary=sentiment_words.values)

    word_matrix = vectorizer.fit_transform(docs)

    return word_matrix.toarray()


project_tests.test_get_bag_of_words(get_bag_of_words)
```

```
Tests Passed
```

Using the `get_bag_of_words` function, we'll generate a bag of words for all the documents.

In [20]:
```python
sentiment_bow_ten_ks = {}

for ticker, ten_ks in ten_ks_by_ticker.items():
    lemma_docs = [' '.join(ten_k['file_lemma']) for ten_k in ten_ks]

    sentiment_bow_ten_ks[ticker] = {
        sentiment: get_bag_of_words(sentiment_df[sentiment_df[sentiment]]['word'], lemma_docs)
        for sentiment in sentiments}


project_helper.print_ten_k_data([sentiment_bow_ten_ks[example_ticker]], sentiments)
```

```
[
  {
    negative: '[[0 0 0 ..., 0 0 0]\n [0 0 0 ..., 0 0 0]\n [0 0 0...
    positive: '[[16  0  0 ...,  0  0  0]\n [16  0  0 ...,  0  0 ...
    uncertainty: '[[0 0 0 ..., 1 1 3]\n [0 0 0 ..., 1 1 3]\n [0 0 0...
    litigious: '[[0 0 0 ..., 0 0 0]\n [0 0 0 ..., 0 0 0]\n [0 0 0...
    constraining: '[[0 0 0 ..., 0 0 2]\n [0 0 0 ..., 0 0 2]\n [0 0 0...
    interesting: '[[2 0 0 ..., 0 0 0]\n [2 0 0 ..., 0 0 0]\n [2 0 0...},
]
```

Jaccard Similarity

Using the bag of words, let's calculate the jaccard similarity on the bag of words and plot it over time. Implement `get_jaccard_similarity` to return the jaccard similarities between each tick in time. Since the input, `bag_of_words_matrix` , is a bag of words for each time period in order, you just need to compute the jaccard similarities for each neighboring bag of words. Make sure to turn the bag of words into a boolean array when calculating the jaccard similarity.

In [21]:
```python
from sklearn.metrics import jaccard_similarity_score


def get_jaccard_similarity(bag_of_words_matrix):
    """
    Get jaccard similarities for neighboring documents

    Parameters
    ----------
    bag_of_words_matrix : 2-d Numpy Ndarray of int
        Bag of words sentiment for each document
        The first dimension is the document.
        The second dimension is the word.

    Returns
    -------
    jaccard_similarities : list of float
        Jaccard similarities for neighboring documents
    """

    # TODO: Implement
    bag_of_words_mat_bool = bag_of_words_matrix.astype(bool)
    jaccard_similarities = []

    for i in range(bag_of_words_mat_bool.shape[0] - 1):
        jaccard_similarities.append(jaccard_similarity_score(bag_of_words_mat_bool[i+1, :], bag_of_words_mat_
bool[i, :]))

    return jaccard_similarities


project_tests.test_get_jaccard_similarity(get_jaccard_similarity)
```

```
Tests Passed
```

Using the `get_jaccard_similarity` function, let's plot the similarities over time.

In [22]:

```
# Get dates for the universe
file_dates = {
    ticker: [ten_k['file_date'] for ten_k in ten_ks]
    for ticker, ten_ks in ten_ks_by_ticker.items()}

jaccard_similarities = {
    ticker: {
        sentiment_name: get_jaccard_similarity(sentiment_values)
        for sentiment_name, sentiment_values in ten_k_sentiments.items()}
    for ticker, ten_k_sentiments in sentiment_bow_ten_ks.items()}


project_helper.plot_similarities(
    [jaccard_similarities[example_ticker][sentiment] for sentiment in sentiments],
    file_dates[example_ticker][1:],
    'Jaccard Similarities for {} Sentiment'.format(example_ticker),
    sentiments)
```

TFIDF

using the sentiment word lists, let's generate sentiment TFIDF from the 10-k documents. Implement `get_tfidf` to generate TFIDF from each document, using sentiment words as the terms. You can ignore words that are not in `sentiment_words` .

In [23]:

```python
from sklearn.feature_extraction.text import TfidfVectorizer


def get_tfidf(sentiment_words, docs):
    """
    Generate TFIDF values from documents for a certain sentiment

    Parameters
    ----------
    sentiment_words: Pandas Series
        Words that signify a certain sentiment
    docs : list of str
        List of documents used to generate bag of words

    Returns
    -------
    tfidf : 2-d Numpy Ndarray of float
        TFIDF sentiment for each document
        The first dimension is the document.
        The second dimension is the word.
    """

    # TODO: Implement
    vectorizer = TfidfVectorizer(vocabulary=sentiment_words.values)

    tfidf = vectorizer.fit_transform(docs)

    return tfidf.toarray()


project_tests.test_get_tfidf(get_tfidf)
```

```
Tests Passed
```

Using the `get_tfidf` function, let's generate the TFIDF values for all the documents.

In [24]:
```
sentiment_tfidf_ten_ks = {}

for ticker, ten_ks in ten_ks_by_ticker.items():
    lemma_docs = [' '.join(ten_k['file_lemma']) for ten_k in ten_ks]

    sentiment_tfidf_ten_ks[ticker] = {
        sentiment: get_tfidf(sentiment_df[sentiment_df[sentiment]]['word'], lemma_docs)
        for sentiment in sentiments}


project_helper.print_ten_k_data([sentiment_tfidf_ten_ks[example_ticker]], sentiments)
```

```
[
  {
    negative: '[[ 0.          0.          0.         ...,  0.   ...
    positive: '[[ 0.22288432  0.          0.         ...,  0.   ...
    uncertainty: '[[ 0.          0.          0.         ...,  0.005...
    litigious: '[[ 0.  0.  0. ...,  0.  0.  0.]\n [ 0.  0.  0. .....
    constraining: '[[ 0.          0.          0.         ...,  0.   ...
    interesting: '[[ 0.01673784  0.          0.         ...,  0.   ...},
]
```

Cosine Similarity

Using the TFIDF values, we'll calculate the cosine similarity and plot it over time. Implement `get_cosine_similarity` to return the cosine similarities between each tick in time. Since the input, `tfidf_matrix` , is a TFIDF vector for each time period in order, you just need to computer the cosine similarities for each neighboring vector.

In [25]:

```
from sklearn.metrics.pairwise import cosine_similarity


def get_cosine_similarity(tfidf_matrix):
    """
    Get cosine similarities for each neighboring TFIDF vector/document

    Parameters
    ----------
    tfidf : 2-d Numpy Ndarray of float
        TFIDF sentiment for each document
        The first dimension is the document.
        The second dimension is the word.

    Returns
    -------
    cosine_similarities : list of float
        Cosine similarities for neighboring documents
    """

    # TODO: Implement
    return list(np.diag(cosine_similarity(tfidf_matrix), k = 1))


project_tests.test_get_cosine_similarity(get_cosine_similarity)
```

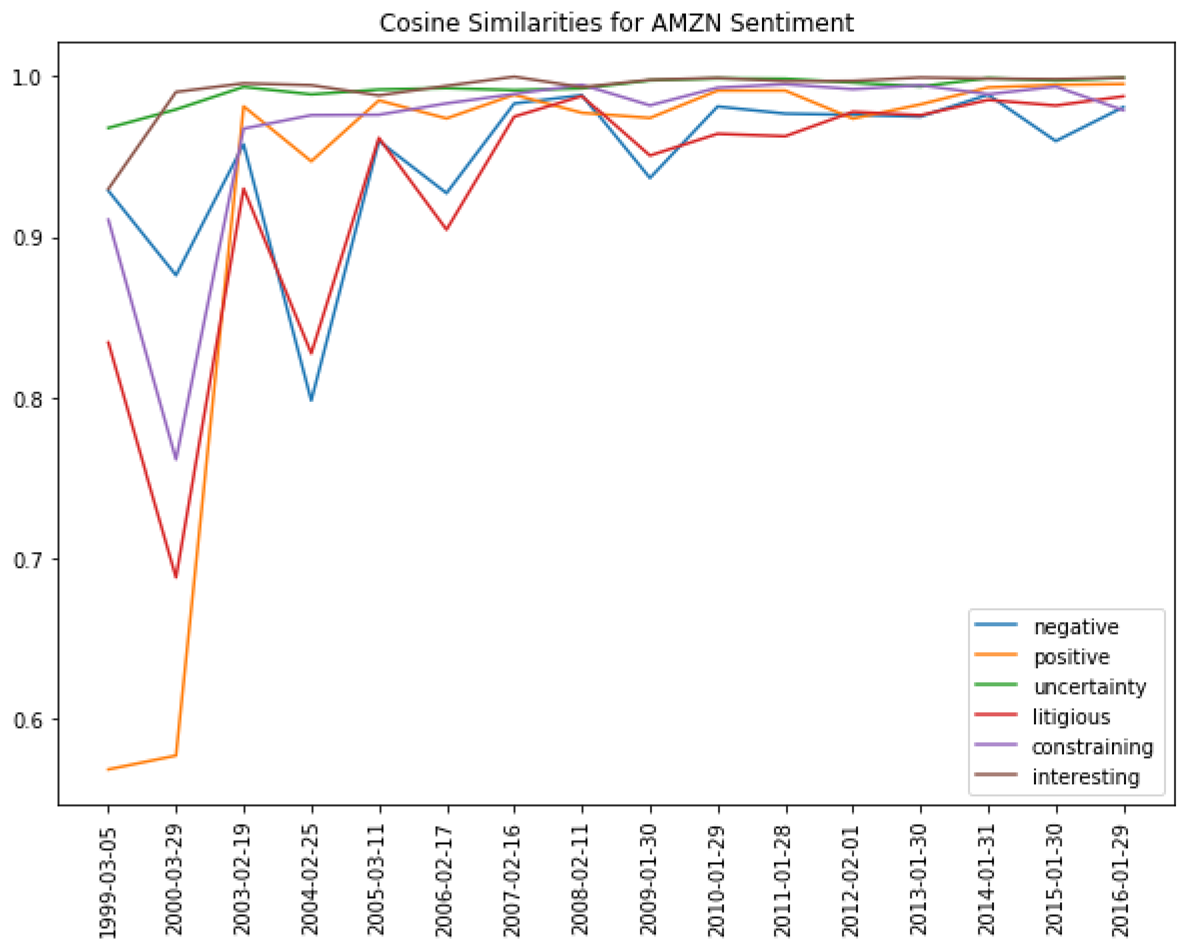
```
Tests Passed
```

Let's plot the cosine similarities over time.

In [26]:

```
cosine_similarities = {
    ticker: {
        sentiment_name: get_cosine_similarity(sentiment_values)
        for sentiment_name, sentiment_values in ten_k_sentiments.items()}
    for ticker, ten_k_sentiments in sentiment_tfidf_ten_ks.items()}


project_helper.plot_similarities(
    [cosine_similarities[example_ticker][sentiment] for sentiment in sentiments],
    file_dates[example_ticker][1:],
    'Cosine Similarities for {} Sentiment'.format(example_ticker),
    sentiments)
```



Evaluate Alpha Factors

Just like we did in project 4, let's evaluate the alpha factors. For this section, we'll just be looking at the cosine similarities, but it can be applied to the jaccard similarities as well.

Price Data

Let's get yearly pricing to run the factor against, since 10-Ks are produced annually.

In [27]:

```
pricing = pd.read_csv('../../data/project_5_yr/yr-quotemedia.csv', parse_dates=['date'])
pricing = pricing.pivot(index='date', columns='ticker', values='adj_close')


pricing
```

Out[27]:

ticker	A	AA	AAAP	AABA	AAC	AADR	AAIT	AAL	AAMC
date									
1962-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1963-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1964-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1965-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1966-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1967-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1968-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1969-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1970-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1971-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1972-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1973-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1974-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1975-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1976-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan
1977-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan

ticker	A	AA	AAAP	AABA	AAC	AADR	AAIT	AAL	AAMC	
date										
1978-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	
1979-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	
1980-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	
1981-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	
1982-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	
1983-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	
1984-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	6.408
1985-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	10.612
1986-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	9.044
1987-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	3.806
1988-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	3.149
1989-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	2.329
1990-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	1.397
1991-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	0.698
1992-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	1.514
1993-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	1.630
1994-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	2.096

ticker	A	AA	AAAP	AABA	AAC	AADR	AAIT	AAL	AAMC	
date										
1995-01-01	nan	nan	nan	nan	nan	nan	nan	nan	nan	2.155
1996-01-01	nan	nan	nan	0.70833333	nan	nan	nan	nan	nan	2.851
1997-01-01	nan	nan	nan	4.32812500	nan	nan	nan	nan	nan	4.718
1998-01-01	nan	nan	nan	29.61250000	nan	nan	nan	nan	nan	4.543
1999-01-01	52.37855258	nan	nan	108.17500000	nan	nan	nan	nan	nan	2.155
2000-01-01	37.09385272	nan	nan	15.03000000	nan	nan	nan	nan	nan	1.863
2001-01-01	19.31590395	nan	nan	8.87000000	nan	nan	nan	nan	nan	2.053
2002-01-01	12.16813872	nan	nan	8.17500000	nan	nan	nan	nan	nan	1.518
2003-01-01	19.81048865	nan	nan	22.51500000	nan	nan	nan	nan	nan	2.795
2004-01-01	16.32807033	nan	nan	37.68000000	nan	nan	nan	nan	nan	2.888
2005-01-01	22.55441748	nan	nan	39.18000000	nan	nan	nan	35.76072005	nan	2.516
2006-01-01	23.61133822	nan	nan	25.54000000	nan	nan	nan	51.85015549	nan	2.758
2007-01-01	24.89183834	nan	nan	23.26000000	nan	nan	nan	14.16371007	nan	1.304
2008-01-01	10.58953275	nan	nan	12.20000000	nan	nan	nan	7.44292854	nan	0.691
2009-01-01	21.05033797	nan	nan	16.78000000	nan	nan	nan	4.66025539	nan	1.192
2010-01-01	28.06937567	nan	nan	16.63000000	nan	28.82785959	nan	9.63825546	nan	1.891
2011-01-01	23.66553928	nan	nan	16.13000000	nan	27.36247977	nan	4.88171380	nan	1.852

ticker	A	AA	AAAP	AABA	AAC	AADR	AAIT	AAL	AAMC	
date										
2012-01-01	28.01179940	nan	nan	19.90000000	nan	30.02536396	27.10695167	12.99864622	82.00000000	2.977
2013-01-01	39.53485221	nan	nan	40.44000000	nan	36.74348283	31.27057728	24.31228275	930.00000000	3.964
2014-01-01	39.43238724	nan	nan	50.51000000	30.92000000	36.88899069	32.90504993	51.89970750	310.12000000	3.947
2015-01-01	40.79862571	nan	31.27000000	33.26000000	19.06000000	38.06921608	30.53000000	41.33893271	17.16000000	4.912
2016-01-01	44.93909238	28.08000000	26.76000000	38.67000000	7.24000000	39.81959334	nan	46.08991196	53.50000000	4.053
2017-01-01	66.65391782	53.87000000	81.62000000	69.85000000	9.00000000	58.83570736	nan	51.80358470	81.60000000	3.378
2018-01-01	61.80000000	46.88000000	81.63000000	73.35000000	9.81000000	52.88000000	nan	37.99000000	67.90000000	2.575

57 rows × 11941 columns

Dict to DataFrame

The alphalens library uses dataframes, so we we'll need to turn our dictionary into a dataframe.

In [28]:
```
cosine_similarities_df_dict = {'date': [], 'ticker': [], 'sentiment': [], 'value': []}


for ticker, ten_k_sentiments in cosine_similarities.items():
    for sentiment_name, sentiment_values in ten_k_sentiments.items():
        for sentiment_values, sentiment_value in enumerate(sentiment_values):
            cosine_similarities_df_dict['ticker'].append(ticker)
            cosine_similarities_df_dict['sentiment'].append(sentiment_name)
            cosine_similarities_df_dict['value'].append(sentiment_value)
            cosine_similarities_df_dict['date'].append(file_dates[ticker][1:][sentiment_values])

cosine_similarities_df = pd.DataFrame(cosine_similarities_df_dict)
cosine_similarities_df['date'] = pd.DatetimeIndex(cosine_similarities_df['date']).year
cosine_similarities_df['date'] = pd.to_datetime(cosine_similarities_df['date'], format='%Y')


cosine_similarities_df.head()
```

Out[28]:

	date	ticker	sentiment	value
0	2016-01-01	AMZN	negative	0.98065125
1	2015-01-01	AMZN	negative	0.95951741
2	2014-01-01	AMZN	negative	0.98838551
3	2013-01-01	AMZN	negative	0.97472377
4	2012-01-01	AMZN	negative	0.97585100

Alphalens Format

In order to use a lot of the alphalens functions, we need to aligned the indices and convert the time to unix timestamp. In this next cell, we'll do just that.

In [29]:
```
import alphalens as al


factor_data = {}
skipped_sentiments = []

for sentiment in sentiments:
    cs_df = cosine_similarities_df[(cosine_similarities_df['sentiment'] == sentiment)]
    cs_df = cs_df.pivot(index='date', columns='ticker', values='value')

    try:
        data = al.utils.get_clean_factor_and_forward_returns(cs_df.stack(), pricing, quantiles=5, bins=None,
periods=[1])
        factor_data[sentiment] = data
    except:
        skipped_sentiments.append(sentiment)

if skipped_sentiments:
    print('\nSkipped the following sentiments:\n{}'.format('\n'.join(skipped_sentiments)))
factor_data[sentiments[0]].head()
```

```
/opt/conda/lib/python3.6/site-packages/statsmodels/compat/pandas.py:56: FutureWarning: The pandas.core.dateto
ols module is deprecated and will be removed in a future version. Please use the pandas.tseries module instea
d.
  from pandas.core import datetools

Dropped 0.0% entries from factor data: 0.0% in forward returns computation and 0.0% in binning phase (set max
_loss=0 to see potentially suppressed Exceptions).
max_loss is 35.0%, not exceeded: OK!
Dropped 0.0% entries from factor data: 0.0% in forward returns computation and 0.0% in binning phase (set max
_loss=0 to see potentially suppressed Exceptions).
max_loss is 35.0%, not exceeded: OK!
Dropped 0.0% entries from factor data: 0.0% in forward returns computation and 0.0% in binning phase (set max
_loss=0 to see potentially suppressed Exceptions).
max_loss is 35.0%, not exceeded: OK!
Dropped 0.0% entries from factor data: 0.0% in forward returns computation and 0.0% in binning phase (set max
_loss=0 to see potentially suppressed Exceptions).
max_loss is 35.0%, not exceeded: OK!
Dropped 0.0% entries from factor data: 0.0% in forward returns computation and 0.0% in binning phase (set max
_loss=0 to see potentially suppressed Exceptions).
max_loss is 35.0%, not exceeded: OK!
Dropped 0.0% entries from factor data: 0.0% in forward returns computation and 0.0% in binning phase (set max
_loss=0 to see potentially suppressed Exceptions).
max_loss is 35.0%, not exceeded: OK!
```

Out[29]:

		1D	factor	factor_quantile
date	**asset**			
1994-01-01	**BMY**	0.53264104	0.44784189	1
	CVX	0.22211880	0.91363233	5
	FRT	0.17159556	0.47730392	3
1995-01-01	**BMY**	0.32152919	0.89403523	1
	CVX	0.28478156	0.91066582	3

Alphalens Format with Unix Time

Alphalen's `factor_rank_autocorrelation` and `mean_return_by_quantile` functions require unix timestamps to work, so we'll also create factor dataframes with unix time.

In [30]:
```
unixt_factor_data = {
    factor: data.set_index(pd.MultiIndex.from_tuples(
        [(x.timestamp(), y) for x, y in data.index.values],
        names=['date', 'asset']))
    for factor, data in factor_data.items()}
```

Factor Returns

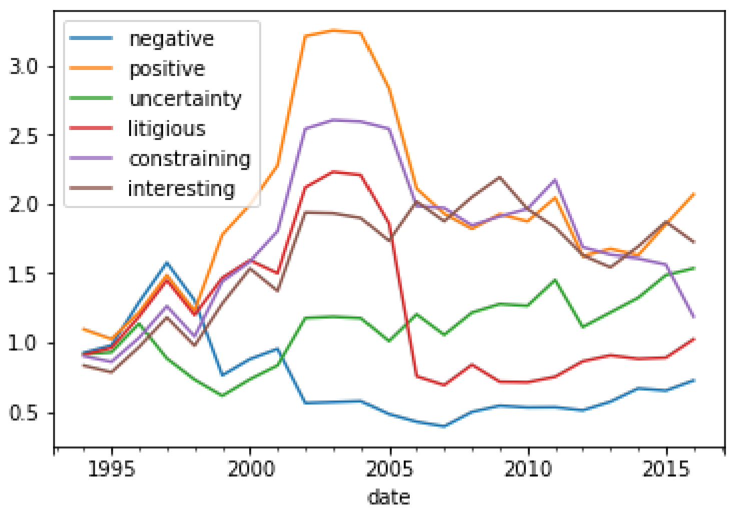
Let's view the factor returns over time. We should be seeing it generally move up and to the right.

In [31]:
```
ls_factor_returns = pd.DataFrame()

for factor_name, data in factor_data.items():
    ls_factor_returns[factor_name] = al.performance.factor_returns(data).iloc[:, 0]

(1 + ls_factor_returns).cumprod().plot()
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f327e1c62b0>



Basis Points Per Day per Quantile

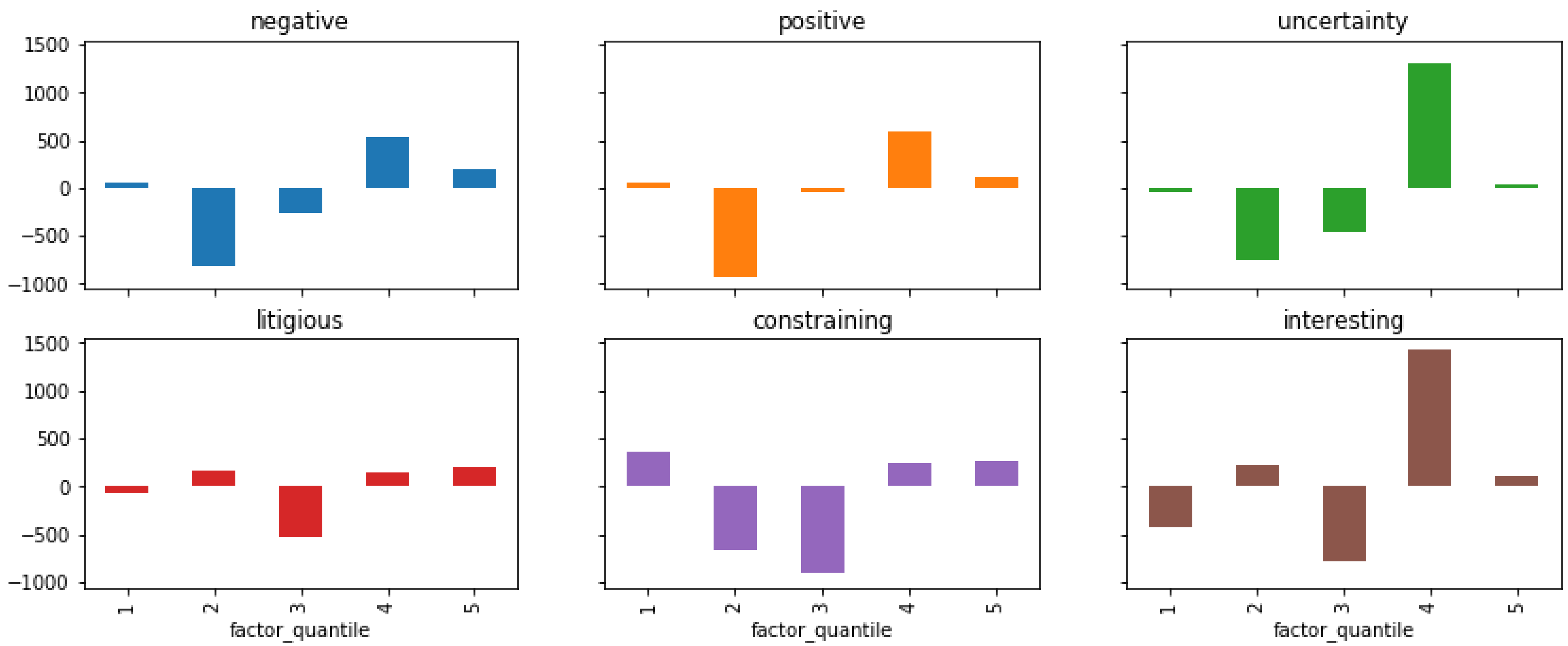
It is not enough to look just at the factor weighted return. A good alpha is also monotonic in quantiles. Let's looks the basis points for the factor returns.

In [32]:
```
qr_factor_returns = pd.DataFrame()

for factor_name, data in unixt_factor_data.items():
    qr_factor_returns[factor_name] = al.performance.mean_return_by_quantile(data)[0].iloc[:, 0]

(10000*qr_factor_returns).plot.bar(
    subplots=True,
    sharey=True,
    layout=(5,3),
    figsize=(14, 14),
    legend=False)
```

Out[32]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f327e18e588>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f327e252358>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f327e3e2ef0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f327e2e6898>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f327e352ef0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f327e352240>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f327e3042b0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f327e43ea90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f327e2af320>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f327e3d5f60>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f327e2f49e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f327e3537f0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f327e30f9e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f327e266e80>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f327e723668>]], dtype=object)
```



Turnover Analysis

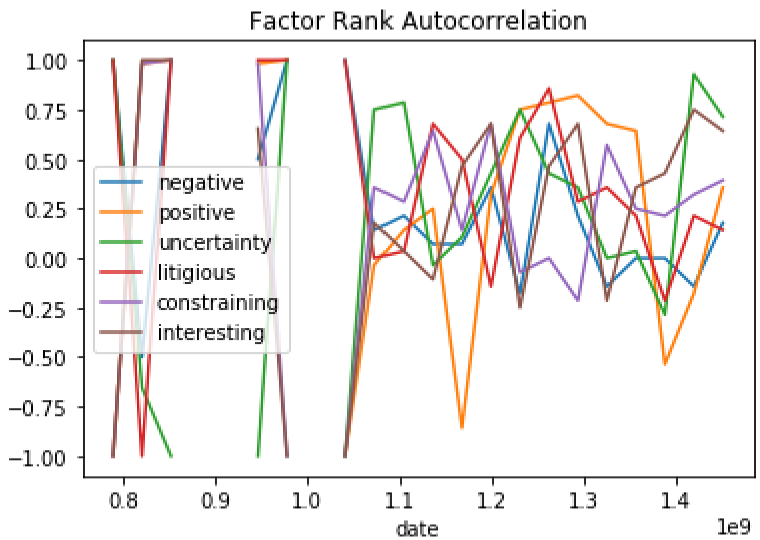
Without doing a full and formal backtest, we can analyze how stable the alphas are over time. Stability in this sense means that from period to period, the alpha ranks do not change much. Since trading is costly, we always prefer, all other things being equal, that the ranks do not change significantly per period. We can measure this with the **Factor Rank Autocorrelation (FRA)**.

In [33]:
```
ls_FRA = pd.DataFrame()

for factor, data in unixt_factor_data.items():
    ls_FRA[factor] = al.performance.factor_rank_autocorrelation(data)

ls_FRA.plot(title="Factor Rank Autocorrelation")
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7f327e6bb940>



Sharpe Ratio of the Alphas

The last analysis we'll do on the factors will be sharpe ratio. Let's see what the sharpe ratio for the factors are. Generally, a Sharpe Ratio of near 1.0 or higher is an acceptable single alpha for this universe.

In [34]:
```
daily_annualization_factor = np.sqrt(252)

(daily_annualization_factor * ls_factor_returns.mean() / ls_factor_returns.std()).round(2)
```

Out[34]:
```
negative       0.39000000
positive       4.17000000
uncertainty    3.05000000
litigious      1.97000000
constraining   1.92000000
interesting    3.49000000
dtype: float64
```

That's it! You've successfully done sentiment analysis on 10-ks!

Submission

Now that you're done with the project, it's time to submit it. Click the submit button in the bottom right. One of our reviewers will give you feedback on your project with a pass or not passed grade. You can continue to the next section while you wait for feedback.