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 and Numpy. 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 [2]:
```
import sys
!{sys.executable} -m pip install -r requirements.txt
```

```
Collecting alphalens==0.3.2 (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/a5/dc/2f9cd107d0d4c
f6223d37d81ddfbbdbf0d703d03669b83810fa6b97f32e5/alphalens-0.3.2.tar.gz (1
8.9MB)
    100% |████████████████████████████████| 18.9MB 1.7MB/s eta 0:00:01
26% |████████                        | 5.0MB 33.3MB/s eta 0:00:01
Collecting nltk==3.3.0 (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/50/09/3b1755d528ad9
156ee7243d52aa5cd2b809ef053a0f31b53d92853dd653a/nltk-3.3.0.zip (1.4MB)
    100% |████████████████████████████████| 1.4MB 12.3MB/s ta 0:00:01   5%
|█                               | 71kB 13.3MB/s eta 0:00:01
Collecting numpy==1.13.3 (from -r requirements.txt (line 3))
  Downloading https://files.pythonhosted.org/packages/57/a7/e3e6bd9d59512
5e1abbe162e323fd2d06f6f6683185294b79cd2cdb190d5/numpy-1.13.3-cp36-cp36m-m
anylinux1_x86_64.whl (17.0MB)
```

```
    100% |████████████████████████████████| 17.0MB 1.8MB/s eta 0:00:01
6% |██                              | 1.1MB 30.5MB/s eta 0:00:01    96% |
███████████████████████████████ | 16.4MB 28.7MB/s eta 0:00:01
Collecting ratelimit==2.2.0 (from -r requirements.txt (line 4))
  Downloading https://files.pythonhosted.org/packages/b5/73/956d739706da2
f74891ba46391381ce7e680dce27cce90df7c706512d5bf/ratelimit-2.2.0.tar.gz
Requirement already satisfied: requests==2.18.4 in /opt/conda/lib/python
3.6/site-packages (from -r requirements.txt (line 5)) (2.18.4)
Requirement already satisfied: scikit-learn==0.19.1 in /opt/conda/lib/pyt
hon3.6/site-packages (from -r requirements.txt (line 6)) (0.19.1)
Requirement already satisfied: six==1.11.0 in /opt/conda/lib/python3.6/si
te-packages (from -r requirements.txt (line 7)) (1.11.0)
Collecting tqdm==4.19.5 (from -r requirements.txt (line 8))
  Downloading https://files.pythonhosted.org/packages/71/3c/341b4fa23cb3a
bc335207dba057c790f3bb329f6757e1fcd5d347bcf8308/tqdm-4.19.5-py2.py3-none-
any.whl (51kB)
    100% |████████████████████████████████| 61kB 13.0MB/s ta 0:00:01
Requirement already satisfied: matplotlib>=1.4.0 in /opt/conda/lib/python
3.6/site-packages (from alphalens==0.3.2->-r requirements.txt (line 1)) (
2.1.0)
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: scipy>=0.14.0 in /opt/conda/lib/python3.6/
site-packages (from alphalens==0.3.2->-r requirements.txt (line 1)) (0.1
9.1)
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: statsmodels>=0.6.1 in /opt/conda/lib/pytho
n3.6/site-packages (from alphalens==0.3.2->-r requirements.txt (line 1))
(0.8.0)
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: chardet<3.1.0,>=3.0.2 in /opt/conda/lib/py
thon3.6/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/py
thon3.6/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/pytho
n3.6/site-packages (from requests==2.18.4->-r requirements.txt (line 5))
(2017.11.5)
Requirement already satisfied: python-dateutil>=2.0 in /opt/conda/lib/pyt
hon3.6/site-packages (from matplotlib>=1.4.0->alphalens==0.3.2->-r requir
ements.txt (line 1)) (2.6.1)
Requirement already satisfied: pytz in /opt/conda/lib/python3.6/site-pack
ages (from matplotlib>=1.4.0->alphalens==0.3.2->-r requirements.txt (line
1)) (2017.3)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.6/s
ite-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 i
```

```
n /opt/conda/lib/python3.6/site-packages (from matplotlib>=1.4.0->alphale
ns==0.3.2->-r requirements.txt (line 1)) (2.2.0)
Requirement already satisfied: simplegeneric>0.8 in /opt/conda/lib/python
3.6/site-packages (from IPython>=3.2.3->alphalens==0.3.2->-r requirement
s.txt (line 1)) (0.8.1)
Requirement already satisfied: pickleshare in /opt/conda/lib/python3.6/si
te-packages (from IPython>=3.2.3->alphalens==0.3.2->-r requirements.txt (
line 1)) (0.7.4)
Requirement already satisfied: pexpect; sys_platform != "win32" in /opt/c
onda/lib/python3.6/site-packages (from IPython>=3.2.3->alphalens==0.3.2->
-r requirements.txt (line 1)) (4.3.1)
Requirement already satisfied: backcall in /opt/conda/lib/python3.6/site-
packages (from IPython>=3.2.3->alphalens==0.3.2->-r requirements.txt (lin
e 1)) (0.1.0)
Requirement already satisfied: pygments in /opt/conda/lib/python3.6/site-
packages (from IPython>=3.2.3->alphalens==0.3.2->-r requirements.txt (lin
e 1)) (2.2.0)
Requirement already satisfied: decorator in /opt/conda/lib/python3.6/site
-packages (from IPython>=3.2.3->alphalens==0.3.2->-r requirements.txt (li
ne 1)) (4.0.11)
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.t
xt (line 1)) (4.3.2)
Requirement already satisfied: setuptools>=18.5 in /opt/conda/lib/python
3.6/site-packages (from IPython>=3.2.3->alphalens==0.3.2->-r requirement
s.txt (line 1)) (38.4.0)
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.15 in /opt/cond
a/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: jedi>=0.10 in /opt/conda/lib/python3.6/sit
e-packages (from IPython>=3.2.3->alphalens==0.3.2->-r requirements.txt (l
ine 1)) (0.10.2)
Requirement already satisfied: ptyprocess>=0.5 in /opt/conda/lib/python3.
6/site-packages (from pexpect; sys_platform != "win32"->IPython>=3.2.3->a
lphalens==0.3.2->-r requirements.txt (line 1)) (0.5.2)
Requirement already satisfied: ipython-genutils in /opt/conda/lib/python
3.6/site-packages (from traitlets>=4.2->IPython>=3.2.3->alphalens==0.3.2-
>-r requirements.txt (line 1)) (0.2.0)
Requirement already satisfied: wcwidth in /opt/conda/lib/python3.6/site-p
ackages (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)
Building wheels for collected packages: alphalens, nltk, ratelimit
  Running setup.py bdist_wheel for alphalens ... done
  Stored in directory: /root/.cache/pip/wheels/77/1e/9a/223b4c94d7f564f25
d94b48ca5b9c53e3034016ece3fd8c8c1
  Running setup.py bdist_wheel for nltk ... done
  Stored in directory: /root/.cache/pip/wheels/d1/ab/40/3bceea46922767e42
986aef7606a600538ca80de6062dc266c
  Running setup.py bdist_wheel for ratelimit ... done
  Stored in directory: /root/.cache/pip/wheels/a6/2a/13/3c6e42757ca0b6873
a60e0697d30f7dd9d521a52874c44f201
Successfully built alphalens nltk ratelimit
tensorflow 1.3.0 requires tensorflow-tensorboard<0.2.0,>=0.1.0, which is
not installed.
moviepy 0.2.3.2 has requirement tqdm==4.11.2, but you'll have tqdm 4.19.5
which is incompatible.
Installing collected packages: numpy, alphalens, nltk, ratelimit, tqdm
```

```
  Found existing installation: numpy 1.12.1
    Uninstalling numpy-1.12.1:
      Successfully uninstalled numpy-1.12.1
  Found existing installation: nltk 3.2.5
    Uninstalling nltk-3.2.5:
      Successfully uninstalled nltk-3.2.5
  Found existing installation: tqdm 4.11.2
    Uninstalling tqdm-4.11.2:
      Successfully uninstalled tqdm-4.11.2
Successfully installed alphalens-0.3.2 nltk-3.3 numpy-1.13.3 ratelimit-2.
2.0 tqdm-4.19.5
```

Load Packages

In [53]:
```python
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 [54]:
```python
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[54]:
```
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 [55]:
```
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 [56]:
```
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 [57]:
```
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()) <=

    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 [58]:
```
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/1018724/000101872417000011/000
1018724-17-000011-index.htm',
  '10-K',
  '2017-02-10'),
 ('https://www.sec.gov/Archives/edgar/data/1018724/000101872416000172/000
1018724-16-000172-index.htm',
  '10-K',
  '2016-01-29'),
 ('https://www.sec.gov/Archives/edgar/data/1018724/000101872415000006/000
1018724-15-000006-index.htm',
  '10-K',
  '2015-01-30'),
 ('https://www.sec.gov/Archives/edgar/data/1018724/000101872414000006/000
1018724-14-000006-index.htm',
  '10-K',
  '2014-01-31'),
 ('https://www.sec.gov/Archives/edgar/data/1018724/000119312513028520/000
1193125-13-028520-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 [59]:
```
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 {
        if (file_type == '10-K'):
            file_url = index_url.replace('-index.htm', '.txt').replace('.

            raw_fillings_by_ticker[ticker][file_date] = sec_api.get(file_


print('Example Document:\n\n{}...'.format(next(iter(raw_fillings_by_ticke
```

```
Downloading AMZN Fillings: 100%|██████████| 22/22 [00:06<00:00,  3.64fill
ing/s]
Downloading BMY Fillings: 100%|██████████| 27/27 [00:23<00:00,  1.13filli
ng/s]
Downloading CNP Fillings: 100%|██████████| 19/19 [00:06<00:00,  2.95filli
ng/s]
Downloading CVX Fillings: 100%|██████████| 25/25 [00:08<00:00,  2.88filli
ng/s]
Downloading FL Fillings: 100%|██████████| 22/22 [00:06<00:00,  3.33fillin
g/s]
Downloading FRT Fillings: 100%|██████████| 29/29 [00:07<00:00,  4.05filli
ng/s]
Downloading HON Fillings: 100%|██████████| 25/25 [00:09<00:00,  2.63filli
ng/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 [60]:
```
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
    start_tag = re.compile(r'<DOCUMENT>')
    end_tag = re.compile(r'</DOCUMENT>')

    start_positions = re.finditer(start_tag, text)
    end_positions = re.finditer(end_tag, text)

    list_of_docs = []

    for start, end in zip(start_positions, end_positions):
        #print(start.span()[0])
        list_of_docs.append(text[start.span()[1]:end.span()[0]])
    #print(text)
    #print(list_of_docs)
    return list_of_docs


project_tests.test_get_documents(get_documents)
```

```
Tests Passed
```

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

In [61]:
```
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 Do
        filling_documents_by_ticker[ticker][file_date] = get_documents(fi


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].it
    for doc_i, doc in enumerate(docs)][:3]))
```

```
Getting Documents from AMZN Fillings: 100%|██████████| 17/17 [00:00<00:0
0, 70.21filling/s]
Getting Documents from BMY Fillings: 100%|██████████| 23/23 [00:00<00:00,
35.45filling/s]
Getting Documents from CNP Fillings: 100%|██████████| 15/15 [00:00<00:00,
34.24filling/s]
Getting Documents from CVX Fillings: 100%|██████████| 21/21 [00:00<00:00,
35.63filling/s]
Getting Documents from FL Fillings: 100%|██████████| 16/16 [00:00<00:00,
64.23filling/s]
Getting Documents from FRT Fillings: 100%|██████████| 19/19 [00:00<00:00,
54.13filling/s]
Getting Documents from HON Fillings: 100%|██████████| 20/20 [00:00<00:00,
41.04filling/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://ww
w.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://ww
w.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 [62]:
```
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
    # I had to remember the solution with a number of non-white-space cha
    # from the previous 'AI for Traging' course content in order to corre
    regex = re.compile(r'<TYPE>[^\s]+')
    doc_type = re.search(regex, doc).group()[len('<TYPE>'):].lower()
    return doc_type


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 [63]:
```
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], ['c
[
  {
    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 [64]:
```python
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 [65]:
```python
for ticker, ten_ks in ten_ks_by_ticker.items():
    for ten_k in tqdm(ten_ks, desc='Cleaning {} 10-Ks'.format(ticker), un
        ten_k['file_clean'] = clean_text(ten_k['file'])


project_helper.print_ten_k_data(ten_ks_by_ticker[example_ticker][:5], ['f
Cleaning AMZN 10-Ks: 100%|██████████| 17/17 [00:35<00:00,  2.07s/10-K]
Cleaning BMY 10-Ks: 100%|██████████| 23/23 [01:13<00:00,  3.18s/10-K]
Cleaning CNP 10-Ks: 100%|██████████| 15/15 [00:55<00:00,  3.67s/10-K]
Cleaning CVX 10-Ks: 100%|██████████| 21/21 [01:52<00:00,  5.37s/10-K]
Cleaning FL 10-Ks: 100%|██████████| 16/16 [00:26<00:00,  1.63s/10-K]
Cleaning FRT 10-Ks: 100%|██████████| 19/19 [00:53<00:00,  2.83s/10-K]
Cleaning HON 10-Ks: 100%|██████████| 20/20 [00:59<00:00,  2.98s/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 [66]:

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


project_tests.test_lemmatize_words(lemmatize_words)
```

```
Tests Passed
```

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

In [67]:
```
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), u
        ten_k['file_lemma'] = lemmatize_words(word_pattern.findall(ten_k[


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

```
Lemmatize AMZN 10-Ks: 100%|██████████| 17/17 [00:04<00:00,  3.7310-K/s]
Lemmatize BMY 10-Ks: 100%|██████████| 23/23 [00:09<00:00,  2.3610-K/s]
Lemmatize CNP 10-Ks: 100%|██████████| 15/15 [00:08<00:00,  1.8210-K/s]
Lemmatize CVX 10-Ks: 100%|██████████| 21/21 [00:09<00:00,  2.2110-K/s]
Lemmatize FL 10-Ks: 100%|██████████| 16/16 [00:04<00:00,  3.9310-K/s]
Lemmatize FRT 10-Ks: 100%|██████████| 19/19 [00:06<00:00,  3.1610-K/s]
Lemmatize HON 10-Ks: 100%|██████████| 20/20 [00:05<00:00,  3.4310-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 [68]:
```
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'.forma
        ten_k['file_lemma'] = [word for word in ten_k['file_lemma'] if wo


print('Stop Words Removed')
```

```
Remove Stop Words for AMZN 10-Ks: 100%|██████████| 17/17 [00:02<00:00, 
7.9610-K/s]
Remove Stop Words for BMY 10-Ks: 100%|██████████| 23/23 [00:04<00:00,  5.
0310-K/s]
Remove Stop Words for CNP 10-Ks: 100%|██████████| 15/15 [00:03<00:00,  3.
9510-K/s]
Remove Stop Words for CVX 10-Ks: 100%|██████████| 21/21 [00:04<00:00,  4.
6610-K/s]
Remove Stop Words for FL 10-Ks: 100%|██████████| 16/16 [00:01<00:00,  8.3
710-K/s]
Remove Stop Words for FRT 10-Ks: 100%|██████████| 19/19 [00:02<00:00,  6.
6010-K/s]
Remove Stop Words for HON 10-Ks: 100%|██████████| 20/20 [00:02<00:00,  7.
5010-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 [69]:
```
import os


sentiments = ['negative', 'positive', 'uncertainty', 'litigious', 'constr

sentiment_df = pd.read_csv(os.path.join('..', '..', 'data', 'project_5_lo
sentiment_df.columns = [column.lower() for column in sentiment_df.columns

# 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[69]:

	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 [70]:
```
from collections import defaultdict, Counter
from sklearn.feature_extraction.text import CountVectorizer


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
    # I used this Internet site to get a better overview on the function:
    # https://scikit-learn.org/stable/modules/feature_extraction.html
    vectorizer = CountVectorizer(vocabulary=sentiment_words)
    X = vectorizer.fit_transform(docs)

    return X.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 [71]:
```
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]]
        for sentiment in sentiments}


project_helper.print_ten_k_data([sentiment_bow_ten_ks[example_ticker]], s
[
  {
    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

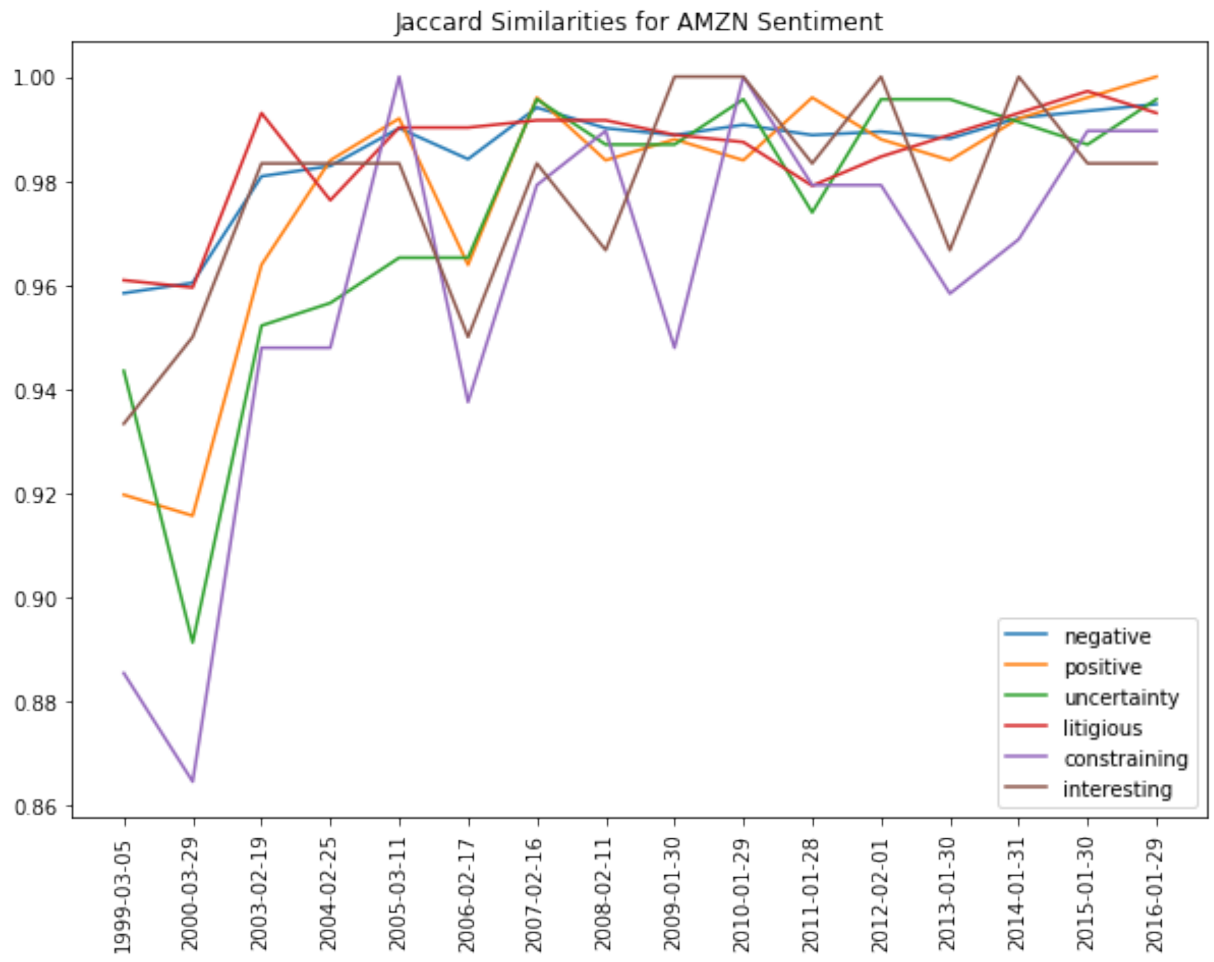
Using the bag of words, let's calculate the jaccard similarity on the bag of words and plot it over time. Implement `get_jaccard_similarity` to return the jaccard similarities between each tick in time. Since the input, `bag_of_words_matrix`, is a bag of words for each time period in order, you just 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 [72]:
```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 : 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
    matrix = bag_of_words_matrix > 0

    jaccard_sim = [jaccard_similarity_score(matrix[i], matrix[i+1]) \
                   for i in range(len(bag_of_words_matrix)-1)]

    return jaccard_sim


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 [73]:
```
# 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 sen
    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 [74]:
```
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

    vec = TfidfVectorizer(vocabulary=sentiment_words)
    X = vec.fit_transform(docs)

    return X.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 [75]:
```
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'
        for sentiment in sentiments}


project_helper.print_ten_k_data([sentiment_tfidf_ten_ks[example_ticker]],
[
  {
    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

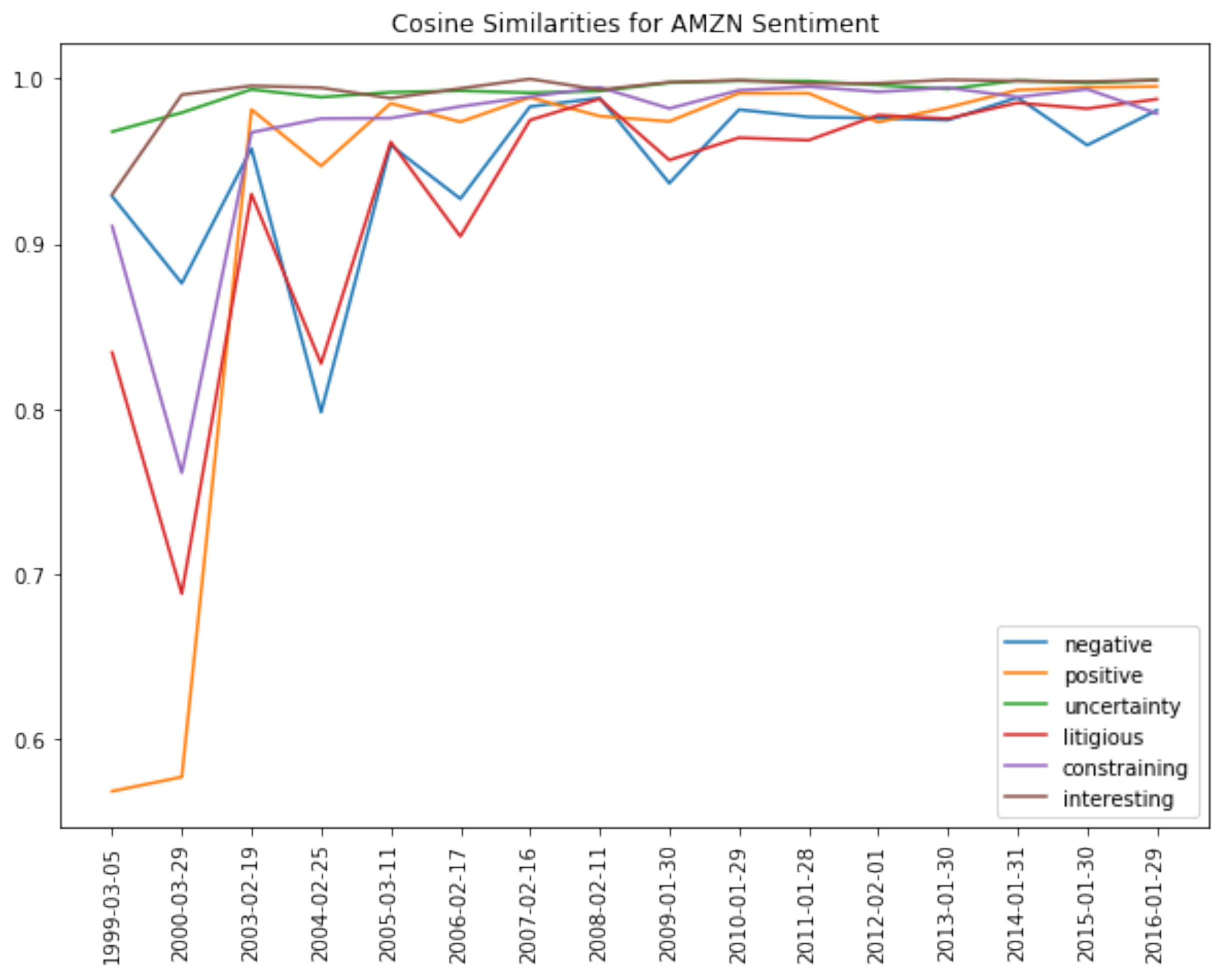
Using the TFIDF values, we'll calculate the cosine similarity and plot it over time. Implement `get_cosine_similarity` to return the cosine similarities between each tick in time. Since the input, `tfidf_matrix`, is a TFIDF vector for each time period in order, you just need to computer the cosine similarities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

In [76]:
```python
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
    # Reshape so that 2D array is passed instead of 1D array
    cosine_sim = [cosine_similarity(tfidf_matrix[i].reshape(1, -1), tfidf
                  for i in range(len(tfidf_matrix)-1)]
    return cosine_sim


project_tests.test_get_cosine_similarity(get_cosine_similarity)
```

```
Tests Passed
```

Let's plot the cosine similarities over time.

In [77]:
```
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 sent
    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 [78]:

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


pricing
```

Out[78]:

ticker	A	AA	AAAP	AABA	AAC	AAD
date						
1962-01-01	nan	nan	nan	nan	nan	na
1963-01-01	nan	nan	nan	nan	nan	na
1964-01-01	nan	nan	nan	nan	nan	na
1965-01-01	nan	nan	nan	nan	nan	na
1966-01-01	nan	nan	nan	nan	nan	na
1967-01-01	nan	nan	nan	nan	nan	na
1968-01-01	nan	nan	nan	nan	nan	na
1969-01-01	nan	nan	nan	nan	nan	na
1970-01-01	nan	nan	nan	nan	nan	na
1971-01-01	nan	nan	nan	nan	nan	na
1972-01-01	nan	nan	nan	nan	nan	na
1973-01-01	nan	nan	nan	nan	nan	na
1974-						

01-01	nan	nan	nan	nan	nan	na
1975-01-01	nan	nan	nan	nan	nan	na
1976-01-01	nan	nan	nan	nan	nan	na
1977-01-01	nan	nan	nan	nan	nan	na
1978-01-01	nan	nan	nan	nan	nan	na
1979-01-01	nan	nan	nan	nan	nan	na
1980-01-01	nan	nan	nan	nan	nan	na
1981-01-01	nan	nan	nan	nan	nan	na
1982-01-01	nan	nan	nan	nan	nan	na
1983-01-01	nan	nan	nan	nan	nan	na
1984-01-01	nan	nan	nan	nan	nan	na
1985-01-01	nan	nan	nan	nan	nan	na
1986-01-01	nan	nan	nan	nan	nan	na
1987-01-01	nan	nan	nan	nan	nan	na
1988-01-01	nan	nan	nan	nan	nan	na
1989-01-01	nan	nan	nan	nan	nan	na
1990-01-01	nan	nan	nan	nan	nan	na
1991-01-01	nan	nan	nan	nan	nan	na
1992-01-01	nan	nan	nan	nan	nan	na
1993-01-01	nan	nan	nan	nan	nan	na
1994-01-01	nan	nan	nan	nan	nan	na
1995-01-01	nan	nan	nan	nan	nan	na

1996-01-01	nan	nan	nan	0.70833333	nan	na
1997-01-01	nan	nan	nan	4.32812500	nan	na
1998-01-01	nan	nan	nan	29.61250000	nan	na
1999-01-01	52.37855258	nan	nan	108.17500000	nan	na
2000-01-01	37.09385272	nan	nan	15.03000000	nan	na
2001-01-01	19.31590395	nan	nan	8.87000000	nan	na
2002-01-01	12.16813872	nan	nan	8.17500000	nan	na
2003-01-01	19.81048865	nan	nan	22.51500000	nan	na
2004-01-01	16.32807033	nan	nan	37.68000000	nan	na
2005-01-01	22.55441748	nan	nan	39.18000000	nan	na
2006-01-01	23.61133822	nan	nan	25.54000000	nan	na
2007-01-01	24.89183834	nan	nan	23.26000000	nan	na
2008-01-01	10.58953275	nan	nan	12.20000000	nan	na
2009-01-01	21.05033797	nan	nan	16.78000000	nan	na
2010-01-01	28.06937567	nan	nan	16.63000000	nan	28.8278595
2011-01-01	23.66553928	nan	nan	16.13000000	nan	27.3624797
2012-01-01	28.01179940	nan	nan	19.90000000	nan	30.0253639
2013-01-01	39.53485221	nan	nan	40.44000000	nan	36.7434828
2014-01-01	39.43238724	nan	nan	50.51000000	30.92000000	36.8889906
2015-01-01	40.79862571	nan	31.27000000	33.26000000	19.06000000	38.0692160
2016-01-01	44.93909238	28.08000000	26.76000000	38.67000000	7.24000000	39.8195933
2017-						

01-01	66.65391782	53.87000000	81.62000000	69.85000000	9.00000000	58.835707
2018-01-01	61.80000000	46.88000000	81.63000000	73.35000000	9.81000000	52.880000

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 [79]:
```
cosine_similarities_df_dict = {'date': [], 'ticker': [], 'sentiment': [],


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_valu
            cosine_similarities_df_dict['ticker'].append(ticker)
            cosine_similarities_df_dict['sentiment'].append(sentiment_nam
            cosine_similarities_df_dict['value'].append(sentiment_value)
            cosine_similarities_df_dict['date'].append(file_dates[ticker]

cosine_similarities_df = pd.DataFrame(cosine_similarities_df_dict)
cosine_similarities_df['date'] = pd.DatetimeIndex(cosine_similarities_df[
cosine_similarities_df['date'] = pd.to_datetime(cosine_similarities_df['d


cosine_similarities_df.head()
```

Out[79]:

	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 [80]:
```python
import alphalens as al


factor_data = {}
skipped_sentiments = []

for sentiment in sentiments:
    cs_df = cosine_similarities_df[(cosine_similarities_df['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(
        factor_data[sentiment] = data
    except:
        skipped_sentiments.append(sentiment)

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

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

Out[80]:

		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.

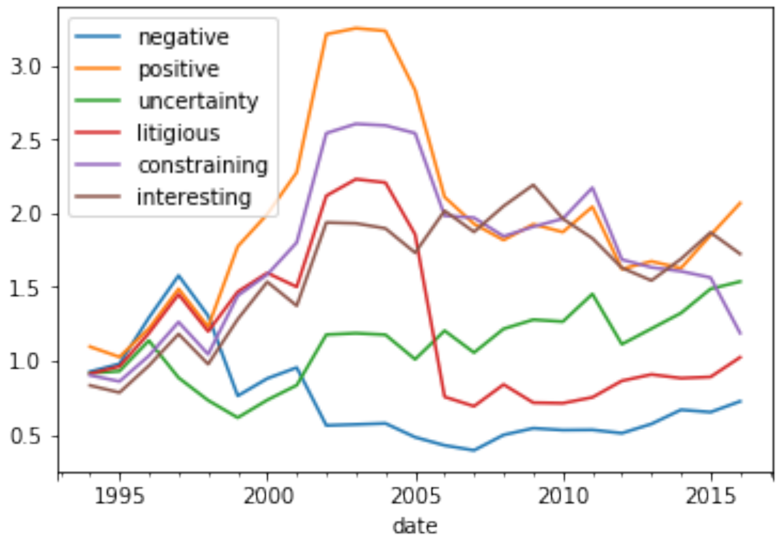
In [81]:
```
unixt_factor_data = {
    factor: data.set_index(pd.MultiIndex.from_tuples(
        [(x.timestamp(), y) for x, y in data.index.values],
        names=['date', 'asset']))
    for factor, data in factor_data.items()}
```

Factor Returns

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

In [82]:
```
ls_factor_returns = pd.DataFrame()

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

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

Out[82]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f379a174ac8>
```

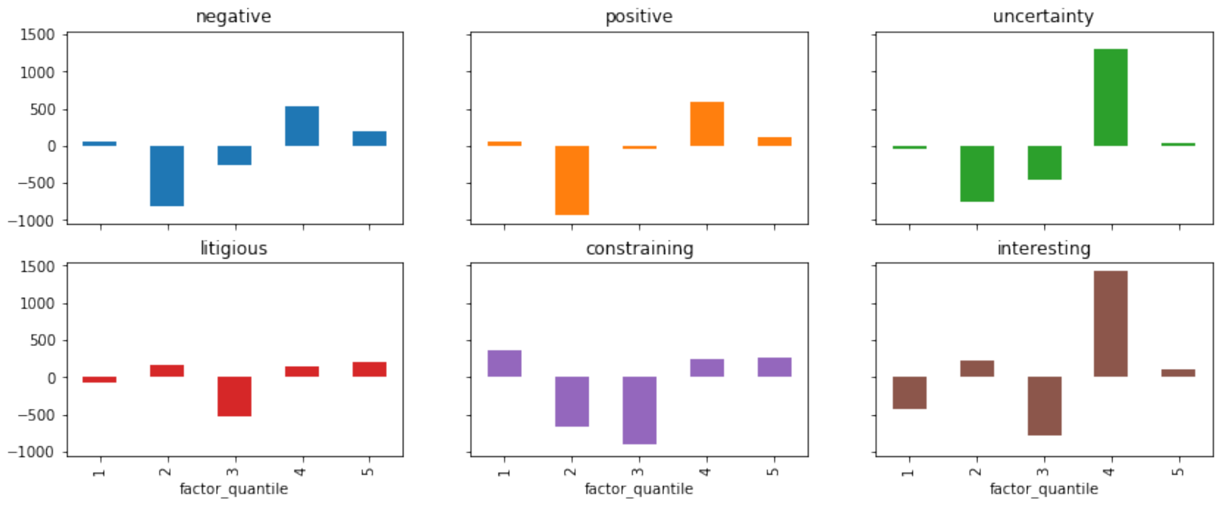


Basis Points Per Day per Quantile

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 [83]:
```
qr_factor_returns = pd.DataFrame()

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

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

Out[83]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f379a1e9dd8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f379a11fb70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3799e0dcc0
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3799df6048>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3799dfdba8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3799dfcba8
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3799e4d208>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3799df2320>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3799e8eba8
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3799e2e0f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3799e75278>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3799ed3eb8
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f3799eae9e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f379a08d518>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f3799e9ccf8
>]], dtype=object)
```

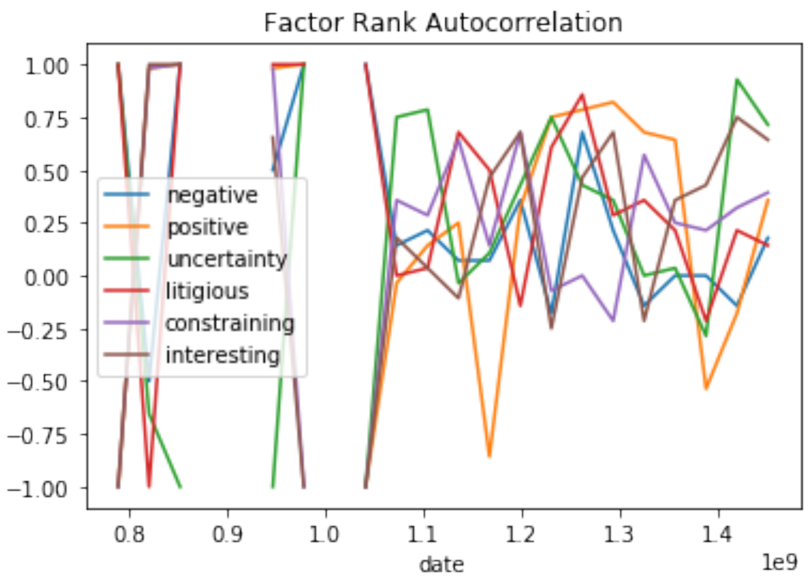


Turnover Analysis

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 [84]:
```
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[84]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3799cb18d0>
```



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 [85]:
```
daily_annualization_factor = np.sqrt(252)

(daily_annualization_factor * ls_factor_returns.mean() / ls_factor_return
```

Out[85]:
```
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.