project_5_starter

March 20, 2023

1 Project 5: NLP on Financial Statements

1.1 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.

1.2 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.

1.2.1 Install Packages

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

```
Collecting alphalens==0.3.2 (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/a5/dc/2f9cd107d0d4cf6223d37d81ddfbbdbf0d70
    100% || 18.9MB 1.8MB/s eta 0:00:01
Collecting nltk==3.3.0 (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/50/09/3b1755d528ad9156ee7243d52aa5cd2b809e
    100% || 1.4MB 15.4MB/s ta 0:00:01
Collecting numpy==1.13.3 (from -r requirements.txt (line 3))
  Downloading https://files.pythonhosted.org/packages/57/a7/e3e6bd9d595125e1abbe162e323fd2d06f6f
    100% || 17.0MB 2.4MB/s eta 0:00:01
Collecting ratelimit==2.2.0 (from -r requirements.txt (line 4))
  Downloading https://files.pythonhosted.org/packages/b5/73/956d739706da2f74891ba46391381ce7e680
Requirement already satisfied: requests==2.18.4 in /opt/conda/lib/python3.6/site-packages (from
Requirement already satisfied: scikit-learn==0.19.1 in /opt/conda/lib/python3.6/site-packages (f
```

```
Requirement already satisfied: six==1.11.0 in /opt/conda/lib/python3.6/site-packages (from -r re
Collecting tqdm==4.19.5 (from -r requirements.txt (line 8))
  Downloading https://files.pythonhosted.org/packages/71/3c/341b4fa23cb3abc335207dba057c790f3bb3
    100% || 61kB 17.7MB/s ta 0:00:01
Requirement already satisfied: matplotlib>=1.4.0 in /opt/conda/lib/python3.6/site-packages (from
Requirement already satisfied: pandas>=0.18.0 in /opt/conda/lib/python3.6/site-packages (from al
Requirement already satisfied: scipy>=0.14.0 in /opt/conda/lib/python3.6/site-packages (from alp
Requirement already satisfied: seaborn>=0.6.0 in /opt/conda/lib/python3.6/site-packages (from al
Requirement already satisfied: statsmodels>=0.6.1 in /opt/conda/lib/python3.6/site-packages (fro
Requirement already satisfied: IPython>=3.2.3 in /opt/conda/lib/python3.6/site-packages (from al
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/conda/lib/python3.6/site-packages (
Requirement already satisfied: idna<2.7,>=2.5 in /opt/conda/lib/python3.6/site-packages (from re
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /opt/conda/lib/python3.6/site-packages (
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.6/site-packages (fro
Requirement already satisfied: python-dateutil>=2.0 in /opt/conda/lib/python3.6/site-packages (f
Requirement already satisfied: pytz in /opt/conda/lib/python3.6/site-packages (from matplotlib>=
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.6/site-packages/cycler-0.1
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /opt/conda/lib/python
Requirement already satisfied: traitlets>=4.2 in /opt/conda/lib/python3.6/site-packages (from IP
Requirement already satisfied: decorator in /opt/conda/lib/python3.6/site-packages (from IPython
Requirement already satisfied: simplegeneric>0.8 in /opt/conda/lib/python3.6/site-packages (from
Requirement already satisfied: pygments in /opt/conda/lib/python3.6/site-packages (from IPython>
Requirement already satisfied: jedi>=0.10 in /opt/conda/lib/python3.6/site-packages (from IPytho
Requirement already satisfied: pickleshare in /opt/conda/lib/python3.6/site-packages (from IPyth
Requirement already satisfied: setuptools>=18.5 in /opt/conda/lib/python3.6/site-packages (from
Requirement already satisfied: backcall in /opt/conda/lib/python3.6/site-packages (from IPython>
Requirement already satisfied: pexpect; sys_platform != "win32" in /opt/conda/lib/python3.6/site
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.15 in /opt/conda/lib/python3.6/site-pa
Requirement already satisfied: ipython-genutils in /opt/conda/lib/python3.6/site-packages (from
Requirement already satisfied: ptyprocess>=0.5 in /opt/conda/lib/python3.6/site-packages (from p
Requirement already satisfied: wcwidth in /opt/conda/lib/python3.6/site-packages (from prompt-to
Building wheels for collected packages: alphalens, nltk, ratelimit
  Running setup.py bdist_wheel for alphalens ... done
  Stored in directory: /home/student/.cache/pip/wheels/77/1e/9a/223b4c94d7f564f25d94b48ca5b9c53e
  Running setup.py bdist_wheel for nltk ... done
  Stored in directory: /home/student/.cache/pip/wheels/d1/ab/40/3bceea46922767e42986aef7606a6005
  Running setup.py bdist_wheel for ratelimit ... done
  Stored in directory: /home/student/.cache/pip/wheels/a6/2a/13/3c6e42757ca0b6873a60e0697d30f7dd
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
  The script tqdm is installed in '/home/student/.local/bin' which is not on PATH.  Consider add
Successfully installed alphalens-0.3.2 nltk-3.3 numpy-1.13.3 ratelimit-2.2.0 tqdm-4.19.5
```

1.2.2 Load Packages

```
In [2]: import nltk
        import numpy as np
        import pandas as pd
        import pickle
        import pprint
        import csv
        import ast

        import project_helper
        import project_tests

        from tqdm import tqdm
        from bs4 import BeautifulSoup
        from collections import defaultdict
```

1.2.3 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
[nltk_data]     /home/student/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /home/student/nltk_data...


Out[3]: True
```

1.3 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'}
```

1.3.1 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.

1.3.2 Upload Stock RSS URL from csv file

```
In [6]: sec_data = {}

        with open('sec_data.csv', 'r') as f:
            reader = csv.reader(f)
            for row in reader:
                ticker, rss_url = row
                sec_data[ticker] = ast.literal_eval(rss_url)
```

```
In [7]: example_ticker = 'AMZN'
        pprint.pprint(sec_data[example_ticker][:5])
```

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

1.3.3 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.

1.3.4 get 10-k data from a csv file

```
In [8]: pd_fillings_by_ticker = {}
        colnames=['ticker', 'file_date', '10-k']

        pd_fillings_by_ticker = pd.read_csv('raw_fillings_by_ticker.csv',
                                            header=None,
                                            names=colnames)
```

```
In [9]: pd_fillings_by_ticker.head()
```

```
Out[9]:   ticker   file_date                                           10-k
        0   AMZN  2017-02-10  <SEC-DOCUMENT>0001018724-17-000011.txt : 20170...
```

```
1   AMZN  2016-01-29  <SEC-DOCUMENT>0001018724-16-000172.txt : 20160...
2   AMZN  2015-01-30  <SEC-DOCUMENT>0001018724-15-000006.txt : 20150...
3   AMZN  2014-01-31  <SEC-DOCUMENT>0001018724-14-000006.txt : 20140...
4   AMZN  2013-01-30  <SEC-DOCUMENT>0001193125-13-028520.txt : 20130...
```

```
In [10]: raw_fillings_by_ticker = defaultdict(dict)

         for i, row in pd_fillings_by_ticker.iterrows():
             raw_fillings_by_ticker[row.ticker][row.file_date] = row['10-k']
```

```
In [11]: print('Example Document:\n\n{}...'.format(next(iter(raw_fillings_by_ticker[example_tick
```

```
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 [5
                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...
```

1.3.5 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 [12]: 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
             pattern = r"<DOCUMENT>(.*?)</DOCUMENT>"
             extracted_docs = re.findall(pattern, text, re.DOTALL)

             return extracted_docs

         project_tests.test_get_documents(get_documents)
```

```
Tests Passed
```

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

```
In [13]: 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 {
                 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 [00:01<00:00,  9.30filling/s]
Getting Documents from BMY Fillings: 100%|| 23/23 [00:04<00:00,  5.03filling/s]
Getting Documents from CNP Fillings: 100%|| 15/15 [00:03<00:00,  4.33filling/s]
Getting Documents from CVX Fillings: 100%|| 21/21 [00:04<00:00,  4.43filling/s]
Getting Documents from FL Fillings: 100%|| 16/16 [00:02<00:00,  7.13filling/s]
Getting Documents from FRT Fillings: 100%|| 19/19 [00:02<00:00,  6.63filling/s]
Getting Documents from HON Fillings: 100%|| 20/20 [00:03<00:00,  6.02filling/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
<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...
```

1.3.6 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 [14]: 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
             pattern = r"<TYPE>[^\n]+"
             match = re.findall(pattern, doc)
             doc_type = str.lower(match[0][len("<TYPE>"):])

             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 [15]: 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', '
```

```
[
  {
    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'},
]
```

1.4 Preprocess the Data

1.4.1 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 [16]: 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 [17]: 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 [00:28<00:00,  1.65s/10-K]
Cleaning BMY 10-Ks: 100%|| 23/23 [01:04<00:00,  2.80s/10-K]
Cleaning CNP 10-Ks: 100%|| 15/15 [00:48<00:00,  3.21s/10-K]
Cleaning CVX 10-Ks: 100%|| 21/21 [01:33<00:00,  4.48s/10-K]
Cleaning FL 10-Ks: 100%|| 16/16 [00:21<00:00,  1.32s/10-K]
Cleaning FRT 10-Ks: 100%|| 19/19 [00:45<00:00,  2.38s/10-K]
Cleaning HON 10-Ks: 100%|| 19/19 [00:45<00:00,  2.39s/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...},
]
```

1.4.2 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 [18]: 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
        lemmatized_words = [WordNetLemmatizer().lemmatize(w, pos='v') for w in words]

        return lemmatized_words


    project_tests.test_lemmatize_words(lemmatize_words)
```

```
Tests Passed
```

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

```
In [ ]: 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 [00:04<00:00,  4.0610-K/s]
Lemmatize BMY 10-Ks: 100%|| 23/23 [00:09<00:00,  2.5110-K/s]
Lemmatize CNP 10-Ks: 100%|| 15/15 [00:07<00:00,  2.0210-K/s]
Lemmatize CVX 10-Ks:  52%|     | 11/21 [00:04<00:04,  2.2410-K/s]
```

1.4.3 Remove Stopwords

```
In [ ]: 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
        ten_k['file_lemma'] = [word for word in ten_k['file_lemma'] if word not in lemma


print('Stop Words Removed')
```

1.5 Analysis on 10ks

1.5.1 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 [ ]: import os


        sentiments = ['negative', 'positive', 'uncertainty', 'litigious', 'constraining', 'inter

        sentiment_df = pd.read_csv(os.path.join('..', '..', 'data', 'project_5_loughran_mcdonald
        sentiment_df.columns = [column.lower() for column in sentiment_df.columns] # Lowercase t

        # 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()
```

1.5.2 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 [ ]: 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
        cnt_vec = CountVectorizer(vocabulary=sentiment_words)
        bag_of_words = cnt_vec.fit_transform(docs).toarray()

        return bag_of_words


    project_tests.test_get_bag_of_words(get_bag_of_words)
```

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

```
In [ ]: 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
                for sentiment in sentiments}


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

1.5.3 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 [ ]: 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
    bag_of_words = bag_of_words_matrix.astype(bool)
    jaccard_similarities = [jaccard_similarity_score(u,v) for u, v in zip(bag_of_words,b

    return jaccard_similarities


project_tests.test_get_jaccard_similarity(get_jaccard_similarity)
```

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

```
In [ ]: # 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)
```

1.5.4 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 [ ]: 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)
            tfidf = vectorizer.fit_transform(docs).toarray()

            return tfidf


        project_tests.test_get_tfidf(get_tfidf)
```

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

```
In [ ]: 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)
```

1.5.5 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 [ ]: 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
            cos_similarity= cosine_similarity(tfidf_matrix[0:], tfidf_matrix[1:])

            return cos_similarity[0].tolist()


        project_tests.test_get_cosine_similarity(get_cosine_similarity)
```

Let's plot the cosine similarities over time.

```
In [ ]: 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)
```

1.6 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 [ ]: 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
```

1.6.1 Dict to DataFrame

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

```
In [ ]: 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_

        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='


        cosine_similarities_df.head()
```

1.6.2 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 [ ]: 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, qua
            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()
```

1.6.3 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 [ ]: 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()}
```

1.6.4 Factor Returns

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

```
In [ ]: 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()
```

1.6.5 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 [ ]: 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].ilo

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

1.6.6 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 [ ]: 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")
```

1.6.7 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 [ ]: daily_annualization_factor = np.sqrt(252)

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

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.