ITCS/DSBA 6100 Big Data Analytics for Competitive Advantage

Preventing Money Laundering

SEC using Big data to monitor financial activity

http://financialservices.mazars.com/wp-content/uploads/2016/04/Big-data-background_-iStock_000039111306_Large.jpg

Team:

Tannu Singh - #801085297

Varun Varia - #801076181

Shailaja Yadav - #801079146

Chirag Nakrani - #801079126

Edward Park - #801097288

Harshini Tanuku - #801061239

User Story for the Project

David W. Henry, National Associate Director, Broker-Dealer Examination Program, Office of Compliance Inspections and Examinations appoints a team to tackle the problem of money laundering and track financial activities. Retail traders, Big banks, hedge funds and other so-called 'big boys' in the financial markets use big data for trade analytics used in high frequency trading, pre-trade decision-support analytics, sentiment measurement, Predictive Analytics etc. Responsibility of the team is to make sure these companies do not misuse the data for their personal gain.

The data collected are the business articles and/or unofficial conversations among Traders, Directors, back office members, marketing personnel, etc. Using Big Data Analytics, we intend to find out what the articles represent and report any anomalies if found.

SEC and Data Analytics?

Insider trading cases often capture headlines and widespread attention as true-life crime stories. Often they include complex schemes, large dollar amounts, celebrities, or well-known Wall Street players. In a "security-based" investigation, the SEC begins its investigation based on suspicious activity in an individual security where the SEC typically notices a news report or receives a referral from the Financial Industry Regulatory Authority (FINRA) related to suspicious trading activity in advance of a merger or acquisition announcement, corporate event, or other significant news causing a sudden or significant movement in the price of the security. The agency then expands its investigation to identify who traded the security in advance of the event and then seeks to find out the motives for the trading activity. To determine who traded the security, the SEC searches literally billions of rows of data in what is commonly referred to as "blue sheets" which are provided to the agency from brokerages and clearing firms.

Story we based around it

The task of our team is to analyze this "blue sheets" as well the articles that are published by big companies and also the materials used by these companies/individuals to make decisions on the trading activities. Our primary purpose is to perform text mining on the data to understand what data is being referred by the trading desk and to ensure that traders only use publicly available information to hold positions in the market
One of our early topic modeling experiments analyzed the information in the tips, complaints, and referrals (also referred to as TCRs) received by the SEC. The goal was to learn whether we could classify themes directly from the data itself and in a way that would enable more efficient triaging of TCRs.

How is Big data used in SEC

You can't run an analysis on an emerging risk unless you know that it is emerging. So this limitation provided motivation for the next phase of our natural language processing efforts. This is when we began applying topic modeling methods, such as latent dirichlet allocation to registrant disclosures and other types of text documents. LDA, as the method is also known, measures the probability of words within documents and across documents, in order to define the unique topics that they represent. This is what the data scientist community calls "unsupervised learning". You don't have to know anything about the content of the documents. No subject matter expertise is needed. LDA extracts insights from the documents, themselves using the data-up approach to define common themes – these are the topics – and report on where, and to what extent, they appear in each document.

Implementation of user story

Text Cleaning

Whenever we get any data its a good practice to clean the data and remove the unwanted stop words, punctuation to clean our data and after the cleaning we can explore the data to make sense out of the document. So here we used different types of technique to clean our data.

Tokenization

Tokenization is the process of breaking up a sequence of strings into pieces such keywords, phrases, symbols and other elements called tokens. We converted our data into sentence token using NLTK sent_tokenize and then to word token using word_tokenize of nltk method. These tokens helped to understand the data and we taken this list of tokens for further cleaning process.

Removed stopwords:

A stop word is a commonly used word (such as "the", "a", "an", "in") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. We removed these stopwords by using already existing stopwords in Python and utilized it to get documents without stop words as these stopwords were not required to take into consideration.

Since stopwords do not give a very good idea of the content present inside the data, we eliminate them. Also, if we build word cloud or perform any topic modelling operations on

the text, the main highlight of every article would be the articles - ‘the’, ‘a’ and ‘and’. After removing the stopwords, making sense of data gets easy.

Removed punctuations and non-English characters:

We removed punctuations and non english characters from the data using “re” (regular function) library of python and taken the clean data into other text file to use it for further cleaning process.

Lemmitised the words using WordNetLemmatizer

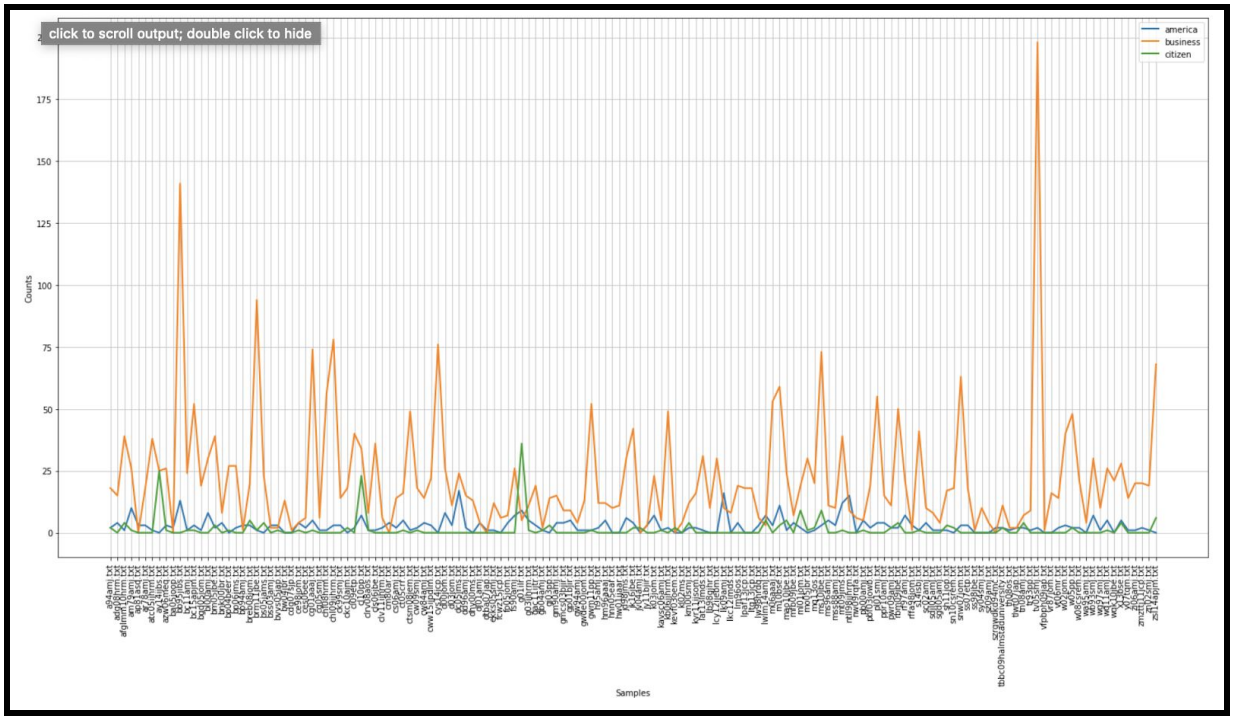
After above cleaning we found out there were many different words which was repeating in different forms, for example, Customer and Customers both were present. Here these both words have same meaning, so we preferred to do lemmatization and combine these words to grab what is important in the document. For this we used nltk method called WordNetLemmatizer.

Data Exploration

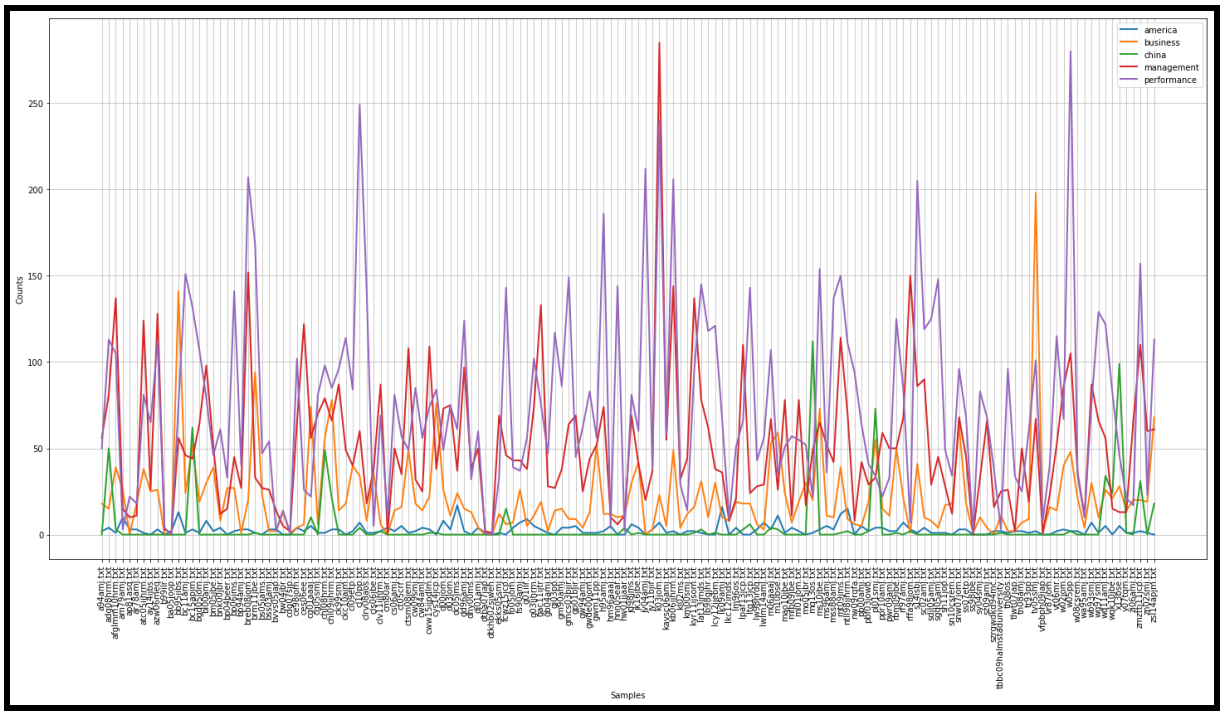
Term Frequency Distribution

In order to get better idea of the data, we needed to perform certain operation on the corpora to make sense out of it. Using PlaintextCorpusReader from nltk library, we load the corpus in jupyter notebook. After that, we iterate each file in the folder in order to perform data exploration. While iterating the data, we found out that some documents had no words or no characters in them which was a noteworthy discovery. We write a short program to display other information about each text by looping over all the values of fileid corresponding to the file identifiers and then computing statistics for each text. For a compact output display, we round each number to the nearest integer, using round().

Our program displays three statistics for each text: average word length, average sentence length, and the number of times each vocabulary item appears in the text on average (our lexical diversity score). This step is important to detect the contents the articles. After performing term frequency distribution, we found out that few documents contained no text. Getting to know average size of a word in the articles and number of words per sentence gives valuable insight to getting to know the data gathered.
Our next step was to implement a plot to visualize the occurence of particular words in the documents. We focused on words ‘america’, ‘business’ and ‘citizens’ in the first attempt and below is the screenshot of occurence of words in each text. Next, we visualize words ‘america’, ‘china’, ‘business’, ‘management’ and ‘performance’.

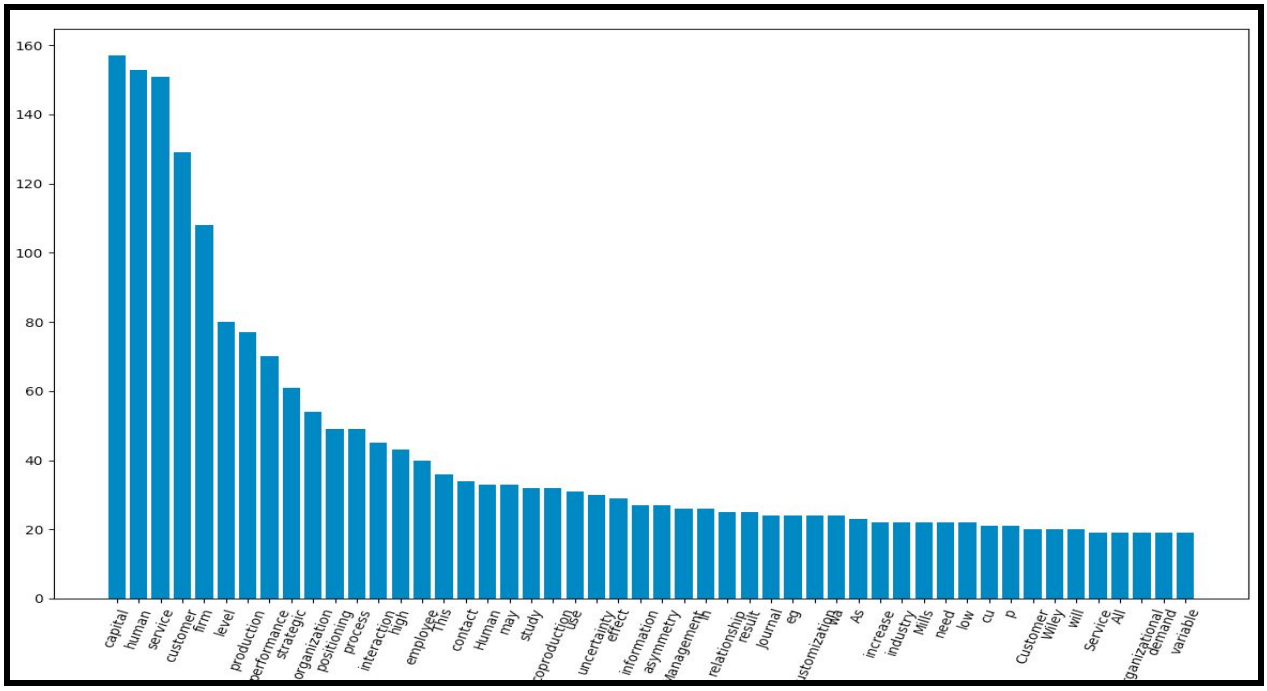


The information that we obtain from the above picture speaks a thousand words, 200 in this case. Using term frequency distribution, we realized that the corpora spoke a lot from the point of view of a business industry and it appeared as if some articles were about exploration of new project overseas or expanding the business to other countries such as China.



Common words:

Common words provide sense about document in a very clear way. We used nltk.probability FreqDist library to generate top 50 words and the visualized it using matplotlib of python to generate bar graph to have proper understanding of each document.



Part of Speech Tagging with NLTK:

NLTK module has the Part of Speech tagging.We try to labeled common words which we generated above in a sentence as nouns, adjectives, verbs, etc. This tagging helped us to get all the words parts of speech which we used to perform chunking in next step.Below is the image attached the way we got in python.

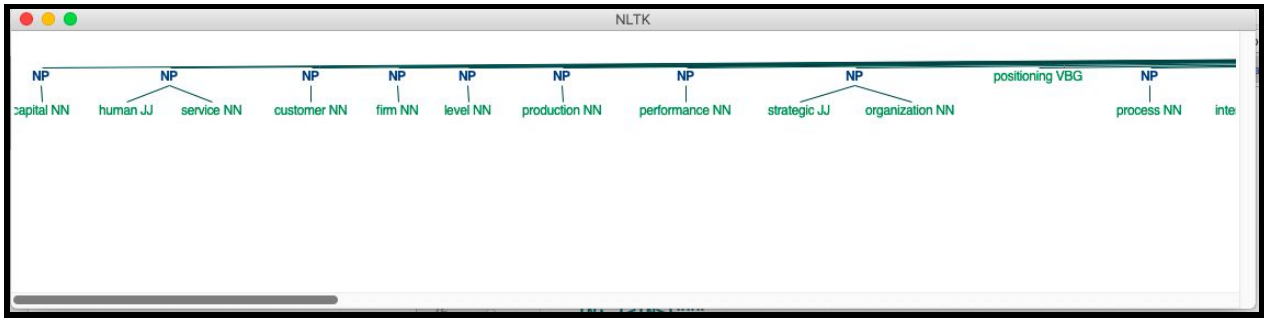
```
/usr/local/bin/python3.7 /Users/shailaja/Desktop/bigdata_project/big_data_prjct.py
[('capital', 'NN'), ('human', 'JJ'), ('service', 'NN'), ('customer', 'NN'), ('firm', 'NN'), ('level', 'NN'),
```

Here's a list of the tags, what they mean, and some examples.

```
POS tag list:

CC  coordinating conjunction
CD  cardinal digit
DT  determiner
EX  existential there (like: "there is" ... think of it like "there exists")
FW  foreign word
IN  preposition/subordinating conjunction
JJ  adjective   'big'
JJR adjective, comparative  'bigger'
JJS adjective, superlative  'biggest'
LS  list marker 1)
MD  modal   could, will
NN  noun, singular 'desk'
NNS noun plural 'desks'
NNP proper noun, singular   'Harrison'
NNPS    proper noun, plural 'Americans'
PDT predeterminer   'all the kids'
POS possessive ending   parent\'s
PRP personal pronoun    I, he, she
PRP$    possessive pronoun  my, his, hers
RB  adverb  very, silently,
RBR adverb, comparative better
RBS adverb, superlative best
RP  particle    give up
TO  to  go 'to' the store.
UH  interjection    errrrrrrrm
VB  verb, base form take
VBD verb, past tense    took
VBG verb, gerund/present participle taking
VBN verb, past participle   taken
VBP verb, sing. present, non-3d take
VBZ verb, 3rd person sing. present  takes
WDT wh-determiner   which
WP  wh-pronoun  who, what
WP$ possessive wh-pronoun   whose
WRB wh-abverb   where, when
```

Chunking with NLTK:

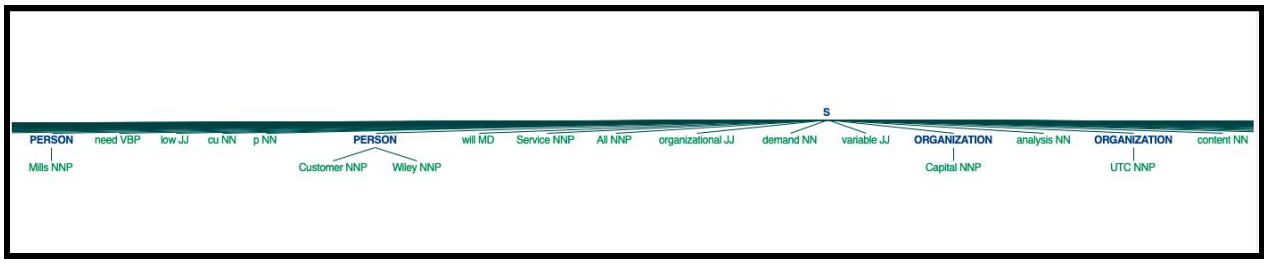
Now that we had the parts of speech, we used method called chunking, and group words into hopefully meaningful chunks. One of the main goals of chunking is to group into what are known as "noun phrases". Our main goal to perform chunking to get clear idea about data and words in some detail way.Below is the figure how it was visible and gave proper understanding of every word.



Named Entity Recognition with NLTK:

One of the most major forms of chunking is called "Named Entity Recognition". The idea is to have the machine immediately be able to pull out "entities" like people, places, things, locations, monetary figures, and more.

Our primary purpose of understanding named entity recognition was to understand the data and get familiar with the content. For Example: "Washington" which appears several times in multiple text could either refer to George Washington/Denzel Washington or it can refer to Washington D.C. Understanding the reference of such words could prove a big difference in detecting and keeping track of trading activities.



Building Word Clouds:

Word clouds present a low-cost alternative for analyzing text from online surveys. Word Clouds work by breaking the text down into component words and counting how frequently they appear in the body of text. Next, the font point size is assigned to words in the cloud based on the frequency that the word appears in the text: the more frequently the word appears, the larger the word is shown in the cloud.
The purpose of building a word cloud in our program is that key words and brand names pop to the surface which helps to provide an overall sense of the text and would help identifying trends and patterns that would otherwise be difficult to see in a tabular or text format.

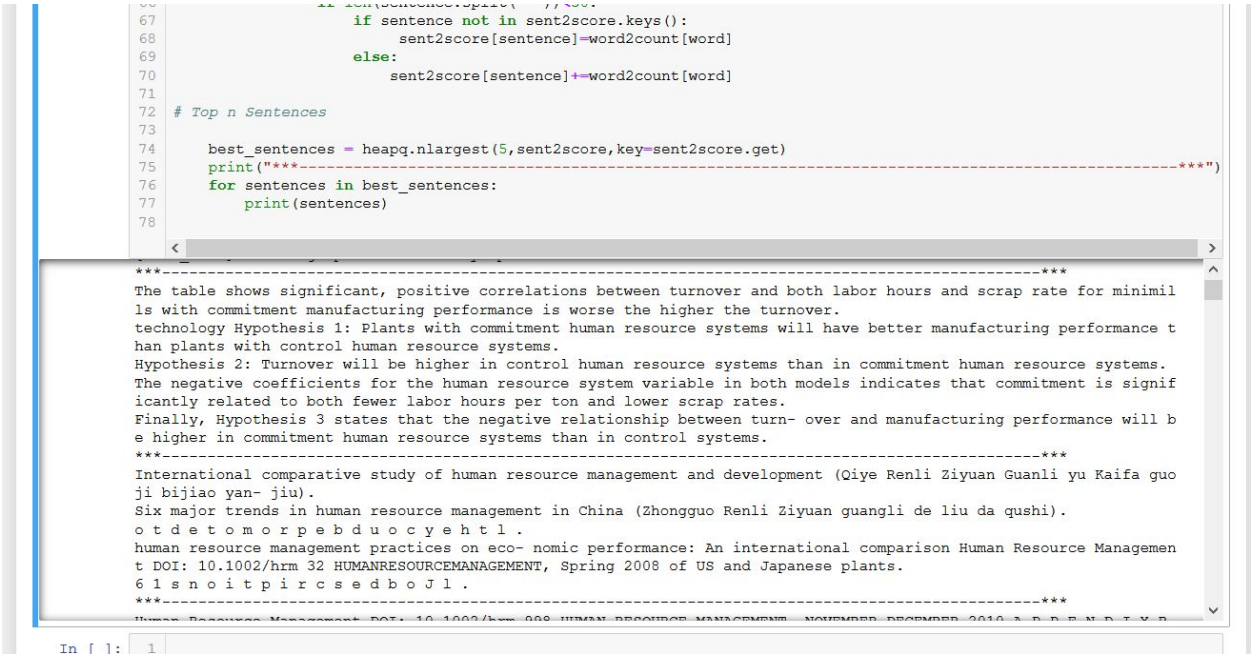
Cosine Similarity

SEC has a record of fraud cases in the past. Cosine similarity can be implemented to check each of the portfolio for any fraud possibilities by comparing the documents with the previous fraud records by comparing pairwise similarity of documents. The documents with the highest pairwise similarity value will most likely be a fraud case and the documents having the least pairwise similarity will most likely be without any fraud possibility.

```
Cosine similarity of document 1 and abs1.txt file:---------->    (0, 1) 0.8627312058451425
  (0, 0)        0.9999999999999377
  (1, 1)        1.0000000000000295
  (1, 0)        0.8627312058451425
Cosine similarity of document 2 and abs1.txt file:---------->    (0, 1) 0.8765265716055046
  (0, 0)        1.0000000000000548
  (1, 1)        0.9999999999999627
  (1, 0)        0.8765265716055046
Cosine similarity of document 3 and abs1.txt file:---------->    (0, 1) 0.9012862328221914
  (0, 0)        0.9999999999998815
  (1, 1)        0.999999999999963
  (1, 0)        0.9012862328221914
Cosine similarity of document 4 and abs1.txt file:---------->    (0, 1) 0.8985270919329914
  (0, 0)        0.9999999999998891
  (1, 1)        0.9999999999999666
  (1, 0)        0.8985270919329914
Cosine similarity of document 5 and abs1.txt file:---------->    (0, 1) 0.876335194323676
  (0, 0)        0.9999999999998997
  (1, 1)        1.0000000000000229
```

Text Summarization

For any suspected fraud portfolio the SEC has the right to intervene into personal records of the management team such as telephone conversation with the family, friends, going through social media accounts. If the data collected for the suspect is too long to skim or to understand then they can use our text summarization service to get the summarized report of the personal data collected. It is evident from the below sample output that our code for text summarization is apt and meaningful.



```
                if sentence not in sent2score.keys():
                    sent2score[sentence]=word2count[word]
                else:
                    sent2score[sentence]+=word2count[word]

# Top n Sentences

    best_sentences = heapq.nlargest(5,sent2score,key=sent2score.get)
    print("***----------------------------------------------------------------------------------------------------***")
    for sentences in best_sentences:
        print(sentences)
```

```
***----------------------------------------------------------------------------------------------------***
The table shows significant, positive correlations between turnover and both labor hours and scrap rate for minimil
ls with commitment manufacturing performance is worse the higher the turnover.
technology Hypothesis 1: Plants with commitment human resource systems will have better manufacturing performance t
han plants with control human resource systems.
Hypothesis 2: Turnover will be higher in control human resource systems than in commitment human resource systems.
The negative coefficients for the human resource system variable in both models indicates that commitment is signif
icantly related to both fewer labor hours per ton and lower scrap rates.
Finally, Hypothesis 3 states that the negative relationship between turn- over and manufacturing performance will b
e higher in commitment human resource systems than in control systems.
***----------------------------------------------------------------------------------------------------***
International comparative study of human resource management and development (Qiye Renli Ziyuan Guanli yu Kaifa guo
ji bijiao yan- jiu).
Six major trends in human resource management in China (Zhongguo Renli Ziyuan guangli de liu da qushi).
o t d e t o m o r p e b d u o c y e h t l .
human resource management practices on eco- nomic performance: An international comparison Human Resource Managemen
t DOI: 10.1002/hrm 32 HUMANRESOURCEMANAGEMENT, Spring 2008 of US and Japanese plants.
6 1 s n o i t p i r c s e d b o J l .
***----------------------------------------------------------------------------------------------------***
```

In [ ]: 1

Topic Modelling

Using topic modelling gave a valuable insight to the materials present in the corpora. After performing topic modelling, we the 5 main topics extracted were management, scrap, performance, human and utc. We also managed to dig top words associated with these topics. Say, management was highly associated with journal or academy, productivity. This way dividing the texts into specific models eliminates a lot of manual work of the SEC to separate and then go through each text to detect any financial fraud or use of information which is not available to public. Using topic modeling if SEC is looking to scrutinize a MNC which deals in offshore business, we can simply pick a relevant topic for scrutiny and analyze the text present in it.

```
C:\Users\singh\Anaconda3\lib\site-packages\sklearn\decomposition\online_lda.py:536: DeprecationWarning: The default v
alue for 'learning_method' will be changed from 'online' to 'batch' in the release 0.20. This warning was introduced
in 0.18.
  DeprecationWarning)
```

```
: 1 import numpy as np
  2 sorting=np.argsort(lda.components_)[:,::-1]
  3 features=np.array(vect.get_feature_names())
```

```
: 1 import mglearn
  2 topic=[]
  3 mglearn.tools.print_topics(topics=range(5), feature_names=features,
  4 sorting=sorting, topics_per_chunk=5, n_words=10)
  5 #rint(topic)
```

```
topic 0       topic 1       topic 2       topic 3       topic 4
--------      --------      --------      --------      --------
model         management    systems       variables     use
work          human         control       evidence      jstor
participation resource      commitment    mills         conditions
firm          turnover      human         resources     terms
strategy      academy       resource      grades        subject
business      journal       employee      research      study
patterns      performance   organizational press         results
decision      strategic     performance   ed            number
industry      labor         turnover      university    employees
customers     arthur        industrial    empirical     production
```

Conclusions -

After performing several text mining operations, we analyzed the data and ran several operations to clean the data and make sure that the data is in correct format for further operations. We managed to solve the mystery of origin of the documents which were generated after converting from pdfs. After performing several visualization, we got a clearer picture of the documents. Our team ran cosine similarity on documents provided to us and with the existing cases of insider trading. Some of these documents were totally different while some had a lot of matching traits with the insider trading documents. The prominent topics of these documents were performance management based on business operations presented in some journal or extracted from it.

Methods taught in class used in the project:-

1. Text cleaning
2. Tokenization
3. Data exploration using NLTK library
4. Cosine similarity
5. Visualization of data - word cloud
6. Topic modelling

Methods used in the project not taught in class:-

1. Chunking with NLTK
2. Named Entity Recognition with NLTK
3. Text Summarization

Appendix 1

Code for the project is included in this section along with relevant screenshots-

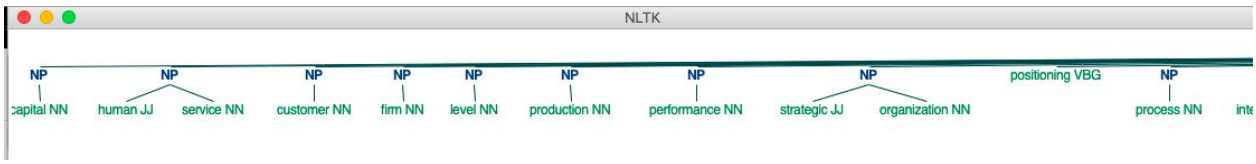
- Text Cleaning:

```
import os
from wordcloud import WordCloud, STOPWORDS
import pandas as pd
import nltk
from nltk import PorterStemmer
from nltk.stem import WordNetLemmatizer
#import numpy as np
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.tokenize import PunktSentenceTokenizer
stopwords = set(STOPWORDS)
path = '/Users/shailaja/Desktop/bigdata_project/Text'
i=0
for itemName in os.listdir(path):
    itemName = os.path.join(path, itemName)
    if os.path.isfile(itemName):
        file= open(itemName,'rt',encoding="utf8")
        train_text=file.read()
        file.close()
        ## word tokenization
        word_tokens=nltk.word_tokenize(train_text)
        print(word_tokens)
```

```
['Strategic', 'Positioning', ',', 'Human', 'Capital', ',', 'and', 'Performance', 'in', 'Service', 'C
':', 'A', 'Customer', 'Interaction', 'Approach', 'Author', '(', 's', ')', ':', 'Bruce', 'C.', 'Skagg
k', 'Youndt', 'Source', ':', 'Strategic', 'Management', 'Journal', ',', 'Vol', '.', '25', ',', 'No',
'Jan.', ',', '2004', ')', ',', 'pp', '.', '85-99', 'Published', 'by', ':', 'Wiley', 'Stable', 'URL',
':', '//www.jstor.org/stable/20142102', 'Accessed', ':', '01-02-2017', '20:10', 'UTC', 'JSTOR', 'is'
rofit', 'service', 'that', 'helps', 'scholars', ',', 'researchers', ',', 'and', 'students', 'discove
',', 'and', 'build', 'upon', 'a', 'wide', 'range', 'of', 'content', 'in', 'a', 'trusted', 'digital',
'We', 'use', 'information', 'technology', 'and', 'tools', 'to', 'increase', 'productivity', 'and', '
w', 'forms', 'of', 'scholarship', '.', 'For', 'more', 'information', 'about', 'JSTOR', ',', 'please'
port', '@', 'jstor.org', '.', 'Your', 'use', 'of', 'the', 'JSTOR', 'archive', 'indicates', 'your', '
f', 'the', 'Terms', '&', 'Conditions', 'of', 'Use', ',', 'available', 'at', 'http', ':', '//about.js
'Wiley', 'is', 'collaborating', 'with', 'JSTOR', 'to', 'digitize', ',', 'preserve', 'and', 'extend',
'Strategic', 'Management', 'Journal', 'This', 'content', 'downloaded', 'from', '152.15.47.133', 'on'
1', 'Feb', '2017', '20:10:15', 'UTC', 'All', 'use', 'subject', 'to', 'http', ':', '//about.jstor.org
ie', 'Management', 'Journal', 'Strat', '.', 'Mgmt', '.', './.', ',', '25', ':', '85-99', '(', '2004'
d', 'online', 'in', 'Wiley', 'InterScience', '(', 'www.interscience.wiley.com', ')', '.', 'DOI', ':'
5', 'STRATEGIC', 'POSITIONING', ',', 'HUMAN', 'CAPITAL', ',', 'AND', 'PERFORMANCE', 'IN', 'SERVICE',
':', 'A', 'CUSTOMER', 'INTERACTION', 'APPROACH', 'BRUCE', 'C.', 'SKAGGS1*', 'and', 'MARK', 'YOUNDT2'
```

```
import re
punctuation = re.compile(r'[-.?!,:;()|0-9]')
post_punctuation = []
for words in word_tokens:
    word = punctuation.sub("", words)
    if len(word) > 0:
        post_punctuation.append(word)

######lemmetization
word_lemmi=[]
lemmi=WordNetLemmatizer()
for word in post_punctuation:
    word_lemmi.append(lemmi.lemmatize(word))
#print(word_lemmi)

### removing stopwords
word_stopword_removed = []
for word in word_lemmi:
    if word in stopwords:
        continue
    else:
        word_stopword_removed.append(word)


words = [word for word in word_stopword_removed if word.isalnum()]
```

```
from nltk.probability import FreqDist

####doesnt include special word
fdist = FreqDist(words)
total_word = fdist.most_common(50)
#print("total_word", total_word)
top_word=[]
for k in total_word:
    top_word.append(k[0])
#print("top_word",top_word)



words_list_tagged=nltk.pos_tag(top_word)
# print(nltk.pos_tag(top_word))
pattern = """NP: {<DT>?<JJ>*<NN>}
VBD: {<VBD>}
IN: {<IN>}"""
NPChunker = nltk.RegexpParser(pattern)
result = NPChunker.parse(words_list_tagged)
result.draw()

words_ne_chunk=nltk.ne_chunk(words_list_tagged,binary=True)
#print(words_ne_chunk)
words_ne_chunk.draw()
```



- Text Exploration:

In [2]:
```
from nltk.corpus import PlaintextCorpusReader
import os.path
import sys

basepath = os.path.dirname('/Users/TheVarunVaria/Downloads/')
corpus_root= os.path.abspath(os.path.join(basepath, "Text"))
wordlists=PlaintextCorpusReader(corpus_root, '.*')
```

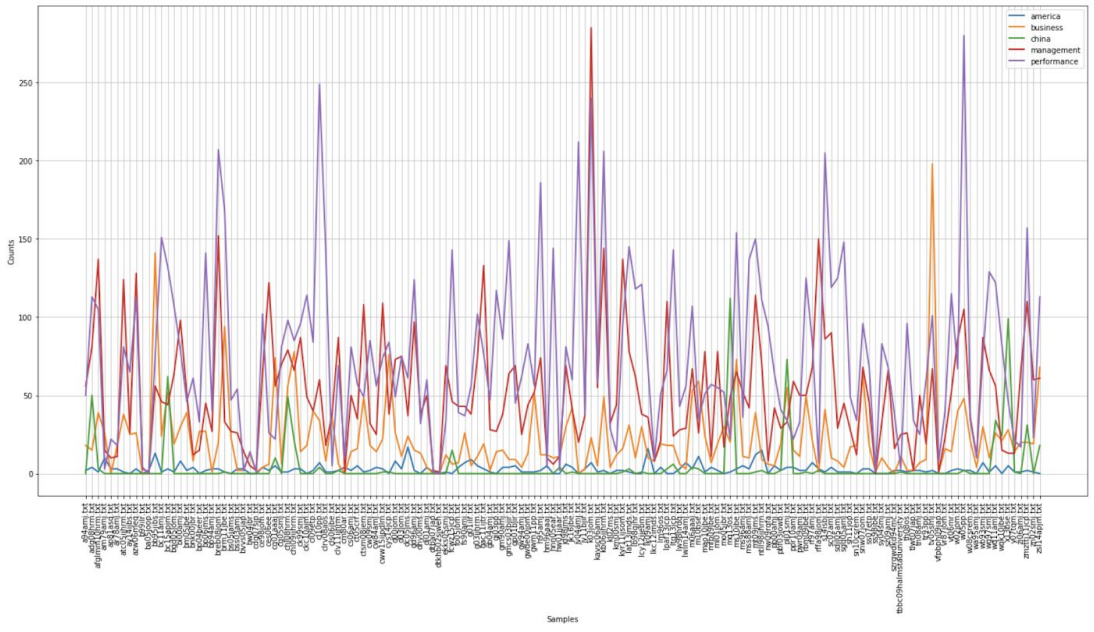
In [3]:
```
for fileid in wordlists.fileids():
    num_char = len(wordlists.raw(fileid))
    num_words = len(wordlists.words(fileid))
    num_sents = len(wordlists.sents(fileid))
    num_vocab = len(set(w.lower() for w in wordlists.words(fileid)))
    print(round(num_words),fileid)
    if(num_words == 0):
          print("Looks like a file with no words", fileid)
    else:
          print(round(num_char/num_words),round(num_words/num_sents),round(num_vocab/num_sents),fileid)
```

```
12875 bgd05jom.txt
5 11 2 bgd05jom.txt
119 bl00aeam.txt
6 7 5 bl00aeam.txt
15517 bl00amj.txt
5 16 3 bl00amj.txt
14108 bm05jbe.txt
5 10 1 bm05jbe.txt
0 bm72rm.txt
Looks like a file with no words bm72rm.txt
10339 bnk00jbr.txt
5 20 3 bnk00jbr.txt
12254 bp04beer.txt
4 6 1 bp04beer.txt
12336 bp06jms.txt
5 16 3 bp06jms.txt
8405 bp94amj.txt
5 12 2 bp94amj.txt
24195 breb08jom.txt
5 10 1 breb08jom.txt
```

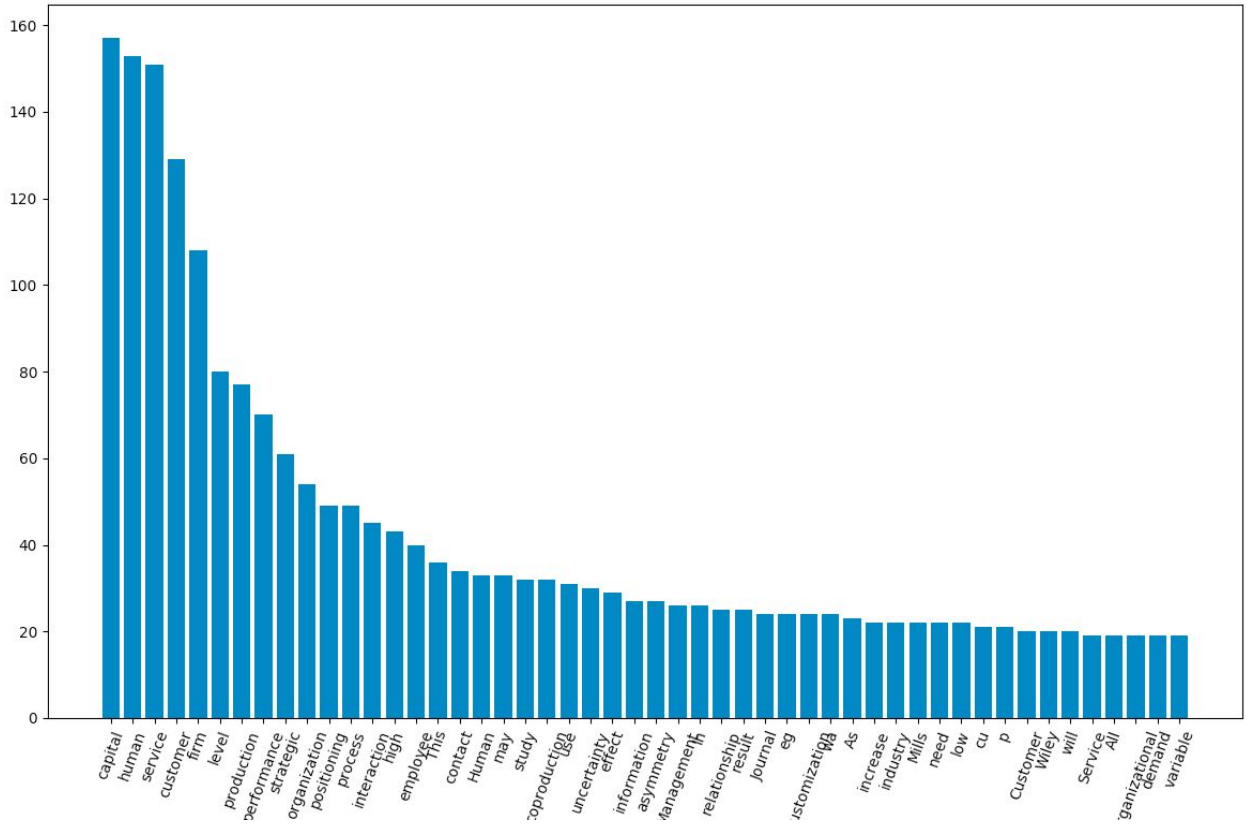
In [11]:
```
import nltk
for fileid in wordlists.fileids():
    news_text = wordlists.words(fileid)
    fdist = nltk.FreqDist(w.lower() for w in news_text)
    modals = ['can','could','would','might','will','washington']
    print (fileid)
    for m in modals:

        print(m + ':',fdist[m], end = ' ')
    print ('\n')
```

```
dd96amj.txt
can: 25 could: 2 would: 12 might: 1 will: 40 washington: 1

dhy00ms.txt
can: 21 could: 2 would: 6 might: 5 will: 8 washington: 5

dj01amj.txt
can: 6 could: 3 would: 4 might: 2 will: 3 washington: 0

dtba07jap.txt
can: 0 could: 0 would: 0 might: 0 will: 0 washington: 1

dtkhb02sjweh.txt
can: 18 could: 4 would: 17 might: 0 will: 13 washington: 0

ekks05smj.txt
can: 36 could: 5 would: 6 might: 16 will: 12 washington: 0

fcwz15jcp.txt
can: 18 could: 7 would: 1 might: 1 will: 12 washington: 0
```

In [13]:
```
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [25, 12]
cfd = nltk.ConditionalFreqDist(
           (target, fileid)
            for fileid in wordlists.fileids()
            for w in wordlists.words(fileid)
            for target in ['america', 'china','business','management','performance']
           if w.lower().startswith(target))
cfd.plot()
```



```
import matplotlib.pyplot as plt
strng=" "
#print(words)
for word in top_word:
    #print("i",i)
    strng+=word+" "
#print(strng)

wordcloud = WordCloud(width=2000, height=2000, stopwords=stopwords, min_font_size=10).generate(strng)

plt.tight_layout()
plt.axis("off")
plt.imshow(wordcloud)
plt.show()

## Bar graph of top words
plt.bar(range(len(total_word)), [val[1] for val in total_word], align='center')
plt.xticks(range(len(total_word)), [val[0] for val in total_word])
plt.xticks(rotation=70)
plt.show()
```


160
140
120
100
80
60
40
20
0
capital
human
service
customer
firm
level
production
performance
strategic
organization
positioning
process
interaction
high
employee
This
contact
Human
may
study
uncertainty
effect
information
asymmetry
relationship
result
Journal
eg
As
increase
industry
Mills
need
low
cu
p
Customer
Wiley
will
Service
All
demand
variable

Appendix 2

Code and output for Cosine Similarity - comparing each document with abs1.txt

[1]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
import glob
txt_files = glob.glob("D:\\Text\\*.txt")
documents = []

j=1

# no need to normalize, since Vectorizer will return normalized tf-idf

abs1= open("abs1.txt", 'rt',encoding="utf8")
absf= abs1.read()
count=1
i=1
for file in txt_files:
    documents.append(absf)

    count=count+1
    file1 = open(file, 'rt',encoding="utf8")
    text = file1.read()
    file1.close()
    documents.append(text)
    tfidf = TfidfVectorizer().fit_transform(documents)
    pairwise_similarity = tfidf * tfidf.T
    print("Cosine similarity of document", i, "and abs1.txt file:----------> ",pairwise_similarity)
    i=i+1
    documents.clear()
print(count)
```

```
Cosine similarity of document 1 and abs1.txt file:---------->    (0, 1) 0.8627312058451425
  (0, 0)        0.9999999999999377
  (1, 1)        1.0000000000000295
  (1, 0)        0.8627312058451425
Cosine similarity of document 2 and abs1.txt file:---------->    (0, 1) 0.8765265716055046
  (0, 0)        1.0000000000000548
  (1, 1)        0.9999999999999627
  (1, 0)        0.8765265716055046
Cosine similarity of document 3 and abs1.txt file:---------->    (0, 1) 0.9012862328221914
  (0, 0)        0.9999999999998815
  (1, 1)        0.999999999999963
  (1, 0)        0.9012862328221914
Cosine similarity of document 4 and abs1.txt file:---------->    (0, 1) 0.8985270919329914
  (0, 0)        0.9999999999998891
  (1, 1)        0.9999999999999666
  (1, 0)        0.8985270919329914
```

Code for Topic Modeling and Intermediate Steps

In [1]:
```
import re
import glob
txt_files = glob.glob("D:\\Text\\*.txt")
for file in txt_files:
    file1 = open(file, 'rt',encoding="utf8")
    text = file1.read()
    file1.close()
    with open(file) as f:
        clean_cont = f.read().splitlines()


    #print(clean_cont)

    shear=[i.replace('\xe2\x80\x9c','') for i in clean_cont ]
    shear=[i.replace('\xe2\x80\x9d','') for i in shear ]
    shear=[i.replace('\xe2\x80\x99s','') for i in shear ]

    shears = [x for x in shear if x != ' ']
    shearss = [x for x in shears if x != '']
    #print(shearss)
    dubby=[re.sub("[^a-zA-Z]+", " ", s) for s in shearss]
    print(dubby)
```

In [3]:
```
#Topic modeling
```

In [4]:
```
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import pandas as pd
import numpy as np
%pylab
%matplotlib inline
```

```
Using matplotlib backend: Qt5Agg
Populating the interactive namespace from numpy and matplotlib
```

```
C:\Users\singh\Anaconda3\lib\site-packages\IPython\core\magics\pylab.py:160: U
ables: ['text', 'f']
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```

In [5]:
```
from sklearn.feature_extraction.stop_words import ENGLISH_STOP_WORDS
```

In [6]:
```
vect=CountVectorizer(ngram_range=(1,1),stop_words='english')
```

In [7]:
```
dtm=vect.fit_transform(dubby)
```

In [8]:
```
pd.DataFrame(dtm.toarray(),columns=vect.get_feature_names())
```

	ability	ables	abound	absence	ac	acad	academy	acceptance	access	accessed	...	wright	www	year
0	0	0	0	0	0	0	1	0	0	0	...	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0
6	0	0	0	0	0	0	0	0	0	1	...	0	0	0
7	0	0	0	0	0	0	0	0	0	0	...	0	1	0
8	0	0	0	0	0	0	1	0	0	0	...	0	0	0
9	0	0	0	0	0	0	0	1	0	0	...	0	1	0
10	0	0	0	0	0	0	0	0	0	0	...	0	0	0

In [9]:
```
lda=LatentDirichletAllocation(n_components=5)
```

In [10]:
```
lda.fit_transform(dtm)
```

```
C:\Users\singh\Anaconda3\lib\site-packages\sklearn\decomposition\online_l
'learning_method' will be changed from 'online' to 'batch' in the release
  DeprecationWarning)
```

Out[10]:
```
array([[0.02857591, 0.0285762 , 0.02857581, 0.02857465, 0.88569743],
       [0.02501538, 0.02531446, 0.02508397, 0.54770511, 0.37688108],
       [0.79998304, 0.0500054 , 0.05000499, 0.05000365, 0.05000292],
       ...,
       [0.02500094, 0.02525626, 0.02500096, 0.02501901, 0.89972283],
       [0.04000052, 0.04001421, 0.04000048, 0.04000035, 0.83998445],
       [0.86662343, 0.03337592, 0.03333353, 0.03333348, 0.03333364]])
```

In [11]:
```
lda_dtf=lda.fit_transform(dtm)
```

```
C:\Users\singh\Anaconda3\lib\site-packages\sklearn\decomposition\onlin
'learning_method' will be changed from 'online' to 'batch' in the rele
  DeprecationWarning)
```

In [12]:
```
import numpy as np
sorting=np.argsort(lda.components_)[:,::-1]
features=np.array(vect.get_feature_names())
```

In [13]:
```
import mglearn
mglearn.tools.print_topics(topics=range(5), feature_names=features,
sorting=sorting, topics_per_chunk=5, n_words=10)
```

topic 0	topic 1	topic 2	topic 3	topic 4
management	scrap	performance	human	utc
use	variables	turnover	resource	content
academy	number	organizational	systems	downloaded
journal	rate	manufacturing	control	fri
jstor	labor	arthur	commitment	results
conditions	hours	employee	management	steel
terms	snell	production	strategic	analysis
subject	work	higher	strategy	research
union	schuler	employees	perspective	mills
productivity	dean	cost	industrial	table