HW 6 - NLP - Kenwan Cheung

You have been provided with a pickle file, containing 100 news articles about some company. Use appropriate topic modeling technique to identify top N most important topics.

read_pickle(directory+news_03.pkl') Present top N most important topics in these news articles Select N to identify relevant topics, but minimize duplication Explain how you selected N Rules and requirements:

Your final output and the code should be contained within Jupyter Notebook

In [17]:
```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [25]:
```
import time
import math
import re
from textblob import TextBlob
import pandas as pd

import nltk as nltk
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer

import string

import warnings
warnings.filterwarnings(action='ignore', category=UserWarning, module='gensim')

import gensim
from gensim import corpora, models
```

In [24]:
```
#nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Boog\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\wordnet.zip.
```

Out[24]:
```
True
```

In [5]:
```python
# import

news = pd.read_pickle('../wk6/news_toyota.pkl')
```

In [6]:
```python
news.head()
```

Out[6]:

	crawled	language	text	title
0	2018-02-02T04:24:51.072+02:00	english	QR Code Link to This Post All maintenance rece...	Dependable truck 03 Toyota Tacoma Double Cab $...
1	2018-02-02T04:27:15.000+02:00	english	0 \nNEW YORK: Automakers reported mixed US car...	US car sales mixed in January; trucks stay strong
2	2018-02-02T04:34:00.008+02:00	english	transmission: automatic 2005 Toyota Camry LE...	2005 TOYOTA CAMRY LE 167300 MILEAGE $2450 (TAL...
3	2018-02-02T04:36:42.006+02:00	english	favorite this post Brand New Toyota Avalon Flo...	Brand New Toyota Avalon Floor Mats (New Britai...
4	2018-02-02T04:38:24.018+02:00	english	more ads by this user QR Code Link to This Pos...	2016 Lexus ES 350 (Coliseum Lexus of Oakland) ...

In [7]:
```python
np.unique(news.language)
```

Out[7]:
```
array(['english'], dtype=object)
```

In [9]:
```python
# remove special characters
news['text_clean'] = news['text'].map(lambda x: re.sub('[^a-zA-Z0-9 @ . , : - _]', '', str(x)))
```

In [10]:
```python
news.head()
```

Out[10]:

	crawled	language	text	title	text_clean
0	2018-02-02T04:24:51.072+02:00	english	QR Code Link to This Post All maintenance rece...	Dependable truck 03 Toyota Tacoma Double Cab $...	QR Code Link to This Post All maintenance rece...
1	2018-02-02T04:27:15.000+02:00	english	0 \nNEW YORK: Automakers reported mixed US car...	US car sales mixed in January; trucks stay strong	0 NEW YORK: Automakers reported mixed US car s...
2	2018-02-02T04:34:00.008+02:00	english	transmission: automatic 2005 Toyota Camry LE...	2005 TOYOTA CAMRY LE 167300 MILEAGE $2450 (TAL...	transmission: automatic 2005 Toyota Camry LE...
3	2018-02-02T04:36:42.006+02:00	english	favorite this post Brand New Toyota Avalon Flo...	Brand New Toyota Avalon Floor Mats (New Britai...	favorite this post Brand New Toyota Avalon Flo...
4	2018-02-02T04:38:24.018+02:00	english	more ads by this user QR Code Link to This Pos...	2016 Lexus ES 350 (Coliseum Lexus of Oakland) ...	more ads by this user QR Code Link to This Pos...

Topic modeling

In [11]:

```
# http://stevenloria.com/finding-important-words-in-a-document-using-tf-idf/

def tf(word, blob):
    return blob.words.count(word) / len(blob.words)
# tf(word, blob) computes "term frequency" which is the number of times a word appears in a document b
lob,
# normalized by dividing by the total number of words in blob. We use TextBlob for breaking up the tex
t into words
# and getting the word counts.


def n_containing(word, bloblist):
    return sum(1 for blob in bloblist if word in blob.words)
# n_containing(word, bloblist) returns the number of documents containing word.
# A generator expression is passed to the sum() function.
```

```python
def idf(word, bloblist):
    return math.log(len(bloblist) / (1 + n_containing(word, bloblist)))
# idf(word, bloblist) computes "inverse document frequency" which measures how common a word is
# among all documents in bloblist. The more common a word is, the lower its idf.
# We take the ratio of the total number of documents to the number of documents containing word,
# then take the log of that. Add 1 to the divisor to prevent division by zero


def tfidf(word, blob, bloblist):
    return tf(word, blob) * idf(word, bloblist)
# tfidf(word, blob, bloblist) computes the TF-IDF score. It is simply the product of tf and idf.
```

In [14]:

```python
bloblist = []
del bloblist[:]

for i  in range(0,len(news)):
    bloblist.append(TextBlob(news['text_clean'].iloc[i]))

len(bloblist)
```

Out[14]: 100

In [18]:

```python
for i, blob in enumerate(bloblist):
# Print top 5 values
    if i == 5:
        break
    print("Top words in news {}".format(i + 1))
    scores = {word: tfidf(word, blob, bloblist) for word in blob.words}
    sorted_words = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    for word, score in sorted_words[:5]:
        print("\tWord: {}, TF-IDF: {}".format(word, round(score, 5)))
```

```
Top words in news 1
        Word: receipts, TF-IDF: 0.21733
        Word: Cash, TF-IDF: 0.21733
        Word: 6477478013, TF-IDF: 0.21733
        Word: sale, TF-IDF: 0.19481
        Word: maintenance, TF-IDF: 0.17883
Top words in news 2
```

```
        Word: And, TF-IDF: 0.06643
        Word: In, TF-IDF: 0.05853
        Word: sales, TF-IDF: 0.04664
        Word: US, TF-IDF: 0.02365
        Word: The, TF-IDF: 0.02218
Top words in news 3
        Word: AUTOMATIC, TF-IDF: 0.18935
        Word: automatic, TF-IDF: 0.15643
        Word: LE, TF-IDF: 0.11506
        Word: cyl, TF-IDF: 0.11506
        Word: VERY, TF-IDF: 0.11506
Top words in news 4
        Word: Mats, TF-IDF: 0.13336
        Word: mats, TF-IDF: 0.13336
        Word: Floor, TF-IDF: 0.08891
        Word: floor, TF-IDF: 0.07969
        Word: Avalon, TF-IDF: 0.06394
Top words in news 5
        Word: included, TF-IDF: 0.0788
        Word: Black, TF-IDF: 0.06732
        Word: Lexus, TF-IDF: 0.06177
        Word: below, TF-IDF: 0.05174
        Word: user, TF-IDF: 0.04396
```

LDA

In [22]:
```
stop = set(stopwords.words('english'))
exclude = set(string.punctuation)
lemma = WordNetLemmatizer()
def clean(doc):
    stop_free = " ".join([i for i in doc.lower().split() if i not in stop])
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)
    normalized = " ".join(lemma.lemmatize(word) for word in punc_free.split())
    return normalized
```

In [19]:
```
news_list = news['text_clean'].tolist()
```

In [26]:
```
news_clean = [clean(doc).split() for doc in news_list]
```

In [29]:
```
# Creating the term dictionary of our courpus, where every unique term is assigned an index.
dictionary = corpora.Dictionary(news_clean)

# Converting list of documents (corpus) into Document Term Matrix using dictionary prepared above.
%time oc_term_matrix = [dictionary.doc2bow(doc) for doc in news_clean]
```

```
Wall time: 15 ms
```

In [30]:
```
# Creating the object for LDA model using gensim library
Lda = gensim.models.ldamodel.LdaModel
numtopics = 3

# Running and Trainign LDA model on the document term matrix.
%time ldamodel = Lda(doc_term_matrix, num_topics=numtopics, id2word = dictionary, passes=50)
```

```
Wall time: 25.1 s
```

Row evaluation

In [31]:
```
print(*ldamodel.print_topics(num_topics=numtopics, num_words=3), sep='\n\n')
```

```
(0, '0.024*"percent" + 0.021*"u" + 0.013*"cent"')

(1, '0.019*"toyota" + 0.007*"japan" + 0.006*"vehicle"')

(2, '0.011*"toyota" + 0.009*"sale" + 0.008*"vehicle"')
```

In [32]:
```
print(*ldamodel.print_topics(num_topics=numtopics, num_words=5), sep='\n\n')
```

```
(0, '0.024*"percent" + 0.021*"u" + 0.013*"cent" + 0.012*"earnings" + 0.012*"per"')

(1, '0.019*"toyota" + 0.007*"japan" + 0.006*"vehicle" + 0.005*"car" + 0.005*"also"')

(2, '0.011*"toyota" + 0.009*"sale" + 0.008*"vehicle" + 0.007*"ford" + 0.007*"car"')
```

In [33]:
```python
print(*ldamodel.print_topics(num_topics=numtopics, num_words=10), sep='\n\n')
```

```
(0, '0.024*"percent" + 0.021*"u" + 0.013*"cent" + 0.012*"earnings" + 0.012*"per" + 0.012*"yield" + 0.0
11*"index" + 0.010*"share" + 0.009*"lower" + 0.009*"investor"')

(1, '0.019*"toyota" + 0.007*"japan" + 0.006*"vehicle" + 0.005*"car" + 0.005*"also" + 0.005*"1" + 0.005
*"2018" + 0.004*"one" + 0.004*"lexus" + 0.003*"canada"')

(2, '0.011*"toyota" + 0.009*"sale" + 0.008*"vehicle" + 0.007*"ford" + 0.007*"car" + 0.007*"year" + 0.0
06*"new" + 0.005*"percent" + 0.005*"unit" + 0.005*"market"')
```

5 topics

In [34]:
```python
# Creating the object for LDA model using gensim library
Lda = gensim.models.ldamodel.LdaModel
numtopics = 5

# Running and Trainign LDA model on the document term matrix.
%time ldamodel = Lda(doc_term_matrix, num_topics=numtopics, id2word = dictionary, passes=50)
```

```
Wall time: 26.2 s
```

In [35]:
```python
print(*ldamodel.print_topics(num_topics=numtopics, num_words=3), sep='\n\n')
```

```
(0, '0.021*"unit" + 0.015*"toyota" + 0.014*"vehicle"')

(1, '0.016*"toyota" + 0.008*"vehicle" + 0.007*"post"')

(2, '0.012*"toyota" + 0.011*"car" + 0.005*"job"')

(3, '0.024*"percent" + 0.021*"u" + 0.013*"earnings"')

(4, '0.017*"sale" + 0.015*"percent" + 0.015*"ford"')
```

In [36]:
```python
print(*ldamodel.print_topics(num_topics=numtopics, num_words=5), sep='\n\n')
```

```
(0, '0.021*"unit" + 0.015*"toyota" + 0.014*"vehicle" + 0.012*"market" + 0.010*"january"')
```

```
(1, '0.016*"toyota" + 0.008*"vehicle" + 0.007*"post" + 0.006*"japan" + 0.006*"car"')

(2, '0.012*"toyota" + 0.011*"car" + 0.005*"job" + 0.005*"company" + 0.005*"state"')

(3, '0.024*"percent" + 0.021*"u" + 0.013*"earnings" + 0.013*"yield" + 0.011*"index"')

(4, '0.017*"sale" + 0.015*"percent" + 0.015*"ford" + 0.010*"year" + 0.010*"toyota"')
```

In [37]:

```
print(*ldamodel.print_topics(num_topics=numtopics, num_words=10), sep='\n\n')
```

```
(0, '0.021*"unit" + 0.015*"toyota" + 0.014*"vehicle" + 0.012*"market" + 0.010*"january" + 0.009*"new"
+ 0.009*"sale" + 0.007*"month" + 0.006*"2018" + 0.006*"share"')

(1, '0.016*"toyota" + 0.008*"vehicle" + 0.007*"post" + 0.006*"japan" + 0.006*"car" + 0.005*"hydrogen"
+ 0.005*"year" + 0.005*"australia" + 0.005*"new" + 0.004*"contact"')

(2, '0.012*"toyota" + 0.011*"car" + 0.005*"job" + 0.005*"company" + 0.005*"state" + 0.005*"d" + 0.004
*"workforce" + 0.004*"said" + 0.004*"alabama" + 0.004*"model"')

(3, '0.024*"percent" + 0.021*"u" + 0.013*"earnings" + 0.013*"yield" + 0.011*"index" + 0.010*"share" +
0.010*"lower" + 0.010*"cent" + 0.009*"per" + 0.009*"investor"')

(4, '0.017*"sale" + 0.015*"percent" + 0.015*"ford" + 0.010*"year" + 0.010*"toyota" + 0.007*"january" +
0.006*"said" + 0.006*"motor" + 0.006*"company" + 0.005*"vehicle"')
```

Discussion

I chose 5 topics with 10 words. We can see quite well what the topic of the news around Toyota represented. Typically sales data that is lost within the smaller model