*# File: TextTechniques.py  
# Name: Gabriel Valenzuela  
# Date: 03/22/2020  
# Course: DSC 550 - Data Mining  
# Purpose: Pt. 2  
# Choose three of the text analysis techniques and apply them to a data file imported into a dataframe  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -***import** pandas **as** pd  
**import** numpy **as** np  
  
controversial\_df = pd.read\_json(**r'C:\Users\Gabe\PycharmProjects\DataMining\_2.2\_BuildYourTextClassifiers\controversial-comments.jsonl'** , lines=**True**)  
controversial\_words\_df = pd.read\_json(**r'C:\Users\Gabe\PycharmProjects\DataMining\_2.2\_BuildYourTextClassifiers\controversial-comments.jsonl'** , lines=**True**)  
controversial\_words\_dict\_df = pd.read\_json(**r'C:\Users\Gabe\PycharmProjects\DataMining\_2.2\_BuildYourTextClassifiers\controversial-comments.jsonl'** , lines=**True**)  
weighing\_df = controversial\_words\_df[1:100]  
tokenize\_df = controversial\_df[1:100]  
dict\_df = controversial\_words\_dict\_df[1:100]  
**'''  
6.4 Tokenizing:  
With tokenizing words, it allows us to separate the sentences within a data frame.  
The main benefit in tokenizing the sentences of a column in a data frame  
is that it allows the data to be processes more quickly.  
'''  
from** nltk.tokenize **import** word\_tokenize  
*# Tokenizing the 'txt' column in the data frame*tokenize\_df[**'txt'**] = tokenize\_df.apply(**lambda** row: word\_tokenize(row[**'txt'**]), axis=1)  
*# Print the first five rows of the data frame*print(tokenize\_df.head())  
  
**'''  
6.9 Weighing Word Importance:  
For this method, it will not only be concerned with the amount of words but with the importance  
of each word to an observation. With each word, the higher the value it has the higher its importance is  
in the data.  
'''  
from** sklearn.feature\_extraction.text **import** TfidfVectorizer  
text\_data = np.array(weighing\_df[**'txt'**])  
tfidf = TfidfVectorizer()  
*# Create the tf-idf feature matrix*feature\_matrix = tfidf.fit\_transform(text\_data)  
print(feature\_matrix)  
*# Shows tf-idf feature matrix as dense matrix*print(feature\_matrix.toarray())  
*# Shows the word of each feature*print(tfidf.vocabulary\_)  
  
**'''  
5.3 Encoding Dictionaries as Features:  
Aids in handling matrices of data that occur during natural language processing. At the same time, it can be used  
in minimizing what is required of the memory.   
'''  
from** sklearn.feature\_extraction **import** DictVectorizer  
*# Counts of words in 'txt' within first 100 rows*series = dict\_df.txt.str.split(expand=**True**).stack().value\_counts()  
*# Create dictionary from word count*data\_dict = series.to\_dict()  
*# Create dictionary vectorizer*dictvectorizer = DictVectorizer(sparse=**False**)  
*# Covert dictionary to feature matrix*features = dictvectorizer.fit\_transform(data\_dict)  
print(features)  
*# Data frame of features*feature\_names = dictvectorizer.get\_feature\_names()  
print(pd.DataFrame(features, columns=feature\_names))