```
import pandas as pd
from collections import Counter
from textblob import TextBlob
import nltk
nltk.download('wordnet')
from textblob import Word
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189

Enter your authorization code:
··········
Mounted at /content/drive
```

```
import os
cwd = os.getcwd()
print (cwd)
```

```
/content
```

```
os.chdir("/content/drive/My Drive/9feb")
cwd = os.getcwd()
print (cwd)
```

```
/content/drive/My Drive/9feb
```

```
from zipfile import ZipFile
with ZipFile('/content/drive/My Drive/9feb/blog-authorship-corpus.zip', 'r') as z:
  z.extractall()
```

```
data = pd.read_csv(r'/content/drive/My Drive/9feb/blogtext.csv')
```

```
data.shape
```

```
data.head(5)
```

```
print (data["text"][2]) # choosing third record
```

```
from bs4 import BeautifulSoup

example1 = BeautifulSoup(data["text"][0])

example1
```

```
print (data["text"][0])
print (example1.get_text())
```

```
import re
letters_only = re.sub("[^a-zA-Z]"," ",example1.get_text() )
print (letters_only)
```

```
lower_case = letters_only.lower()

words = lower_case.split()
```

```
  >>> import nltk
  >>> nltk.download('stopwords')
```

```
from nltk.corpus import stopwords
print(stopwords.words("english") )
```

```
words = [w for w in words if not w in stopwords.words("english")]
print(words)
```

```
from nltk.stem.porter import PorterStemmer
porter_stemmer = PorterStemmer()
porter_stemmer.stem('running')
```

```
from nltk.stem.lancaster import LancasterStemmer
lancaster_stemmer = LancasterStemmer()
porter_stemmer.stem('running')
```

```
from nltk.stem.porter import PorterStemmer
for idx,x in enumerate(words):
    porter_stemmer = PorterStemmer()
    words[idx]= porter_stemmer.stem(x)
print(words)
```

```
for idx, val in enumerate(words):
    print(idx, val)
```

```
   import nltk
nltk.download('punkt')
```

```
 >>> nltk.download('averaged_perceptron_tagger')
```

```
#POS tagging demo
import nltk
text = nltk.word_tokenize("A sample example to test POS tagging with python")

nltk.pos_tag(text)
```

```
 >>> nltk.download('tagsets')
```

```
 nltk.help.upenn_tagset('JJ')
```

```
nltk.help.upenn_tagset('NNP')
```

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
wordnet_lemmatizer = WordNetLemmatizer()
print(wordnet_lemmatizer.lemmatize('is', pos='v'))
print(wordnet_lemmatizer.lemmatize('better', pos='r'))
```

```
sent=" ".join(words)

print(sent)
```

```
data.topic.value_counts()
```

```
data_topic_count= data.topic.value_counts()
```

```
data_topic_count.columns = ['topic', 'topic_count']
```

```
data_topic_count.head()
```

```
data.dtypes
```

```
data.topic = data.topic.astype(str)
```

```
import pandas as pd
from collections import Counter
from textblob import TextBlob
import nltk
nltk.download('wordnet')
from textblob import Word
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

```
import nltk
nltk.download('stopwords')
```

```
from nltk.tokenize import RegexpTokenizer
from nltk.stem import WordNetLemmatizer,PorterStemmer
from nltk.corpus import stopwords
import re
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()
```

```
print (stopwords.words("english"))
```

```
def review_to_words( raw_review ): # above we were working on one record and now we are creat
    # Function to convert a raw review to a string of words
    # The input is a single string (a raw movie review), and
    # the output is a single string (a preprocessed movie review)
    #
    # 1. Remove HTML
    review_text = BeautifulSoup(raw_review).get_text()
    #
    # 2. Remove non-letters
    letters_only = re.sub("[^a-zA-Z]", " ", review_text)
    #
    # 3. Convert to lower case, split into individual words
    words = letters_only.lower().split()
    #
    # 4. In Python, searching a set is much faster than searching
    #   a list, so convert the stop words to a set
    stops = set(stopwords.words("english"))
    #
    # 5. Remove stop words
    meaningful_words = [w for w in words if not w in stops]
    #
    # 6. Join the words back into one string separated by space,
    # and return the result.
    return( " ".join( meaningful_words ))

clean_review = review_to_words( data["text"][0] )
print(clean_review)
```

```
# Get the number of reviews based on the dataframe column size
num_reviews = data["text"].size
num_reviews
```

```
# Initialize an empty list to hold the clean reviews
clean_train_reviews = []
clean_train_reviews
```

```
# Loop over each review; create an index i that goes from 0 to the length
# of the movie review list
for i in range( 0, num_reviews ):
    # Call our function for each one, and add the result to the list of
    # clean reviews
    clean_train_reviews.append( review_to_words( data["text"][i] ) )

clean_train_reviews[10]
```

```
print("Creating the bag of words...\n")
from sklearn.feature_extraction.text import CountVectorizer

# Initialize the "CountVectorizer" object, which is scikit-learn's
# bag of words tool.
vectorizer = CountVectorizer(analyzer = "word",   \
                             tokenizer = None,    \
                             preprocessor = None, \
                             stop_words = None,   \
                             max_features = 5000)

# fit_transform() does two functions: First, it fits the model
# and learns the vocabulary; second, it transforms our training data
# into feature vectors. The input to fit_transform should be a list of
# strings.
train_data_features = vectorizer.fit_transform(clean_train_reviews)

# Numpy arrays are easy to work with, so convert the result to an
# array
train_data_features = train_data_features.toarray()
```

```
print train_data_features.shape


# Take a look at the words in the vocabulary
vocab = vectorizer.get_feature_names()
print vocab


import numpy as np

# Sum up the counts of each vocabulary word
dist = np.sum(train_data_features, axis=0)

# For each, print the vocabulary word and the number of times it
# appears in the training set
for tag, count in zip(vocab, dist):
    print count, tag


from sklearn.ensemble import RandomForestClassifier
from sklearn.cross_validation import cross_val_score
import numpy as np
# Initialize a Random Forest classifier with 100 trees
forest = RandomForestClassifier(n_estimators = 100)
# Fit the forest to the training set, using the bag of words as
# features and the sentiment labels as the response variable
#
# This may take a few minutes to run
print "Training the random forest..."
forest = forest.fit( train_data_features, train["sentiment"] )
# random forest performance through cross vaidation
print forest
print np.mean(cross_val_score(forest,train_data_features,train["sentiment"],cv=10))


# Read the test data
test = pd.read_csv("testData.tsv", header=0, delimiter="\t", \
                   quoting=3 )

# Verify that there are 25,000 rows and 2 columns
print test.shape
```

```
# Create an empty list and append the clean reviews one by one
num_reviews = len(test["review"])
clean_test_reviews = []

print "Cleaning and parsing the test set movie reviews...\n"
for i in xrange(0,num_reviews):
    if( (i+1) % 1000 == 0 ):
        print "Review %d of %d\n" % (i+1, num_reviews)
    clean_review = review_to_words( test["review"][i] )
    clean_test_reviews.append( clean_review )

# Get a bag of words for the test set, and convert to a numpy array
test_data_features = vectorizer.transform(clean_test_reviews)
test_data_features = test_data_features.toarray()

# Use the random forest to make sentiment label predictions
result = forest.predict(test_data_features)
print result

# Copy the results to a pandas dataframe with an "id" column and
# a "sentiment" column
output = pd.DataFrame( data={"id":test["id"], "sentiment":result} )

# Use pandas to write the comma-separated output file
output.to_csv( "Bag_of_Words_model.csv", index=False, quoting=3 )
```

**do till here **

```
train_2000= data.groupby("topic").filter(lambda x: len(x) > 2000 )
train_2000.shape
```

```
train_5000= data.groupby("topic").filter(lambda x: len(x) < 5000 )
train_5000.shape
```

```
train_5000.topic.value_counts()
```

```
train_5000.to_csv(r'/content/drive/My Drive/9feb/reduced_blogtext.csv', index= False)
```

```
train_5000.isnull().sum()
```

```
train_5000['text']
```

```
clean_review = review_to_words( train_5000['text'][4] )
print (clean_review)
```

```
train_5000.columns
```

```
train_5000.dtypes
```

```
train_5000.age = train_5000.age.astype('str')
```

```
train_5000.dtypes
```

```
train_5000["labels"] = train_5000["gender"] + "," + train_5000["age"] + "," +  train_5000["to
train_5000
```

```
train_5000 = train_5000.drop('id', axis=1)
```

```
train_5000 = train_5000.drop('date', axis=1)
```

```
train_5000.head(1)
```

```
train_5000 = train_5000.drop('gender', axis=1)
```

```
train_5000 = train_5000.drop('age', axis=1)
```

```
train_5000 = train_5000.drop('topic', axis=1)
```

```
train_5000 = train_5000.drop('sign', axis=1)
```

```
train_5000
```

```
# Get the number of reviews based on the dataframe column size
num_reviews = train_5000["text"].size
num_reviews
```

```
# Initialize an empty list to hold the clean reviews
clean_train_reviews = []
clean_train_reviews
```

```
 train_5000["text"][4]
```

```
# Loop over each review; create an index i that goes from 0 to the length
# of the movie review list
for i in range( 0, num_reviews ):
    # Call our function for each one, and add the result to the list of
    # clean reviews
    clean_train_reviews.append(review_to_words(train_5000["text"][i] ) )
```

train_5000

train_5000