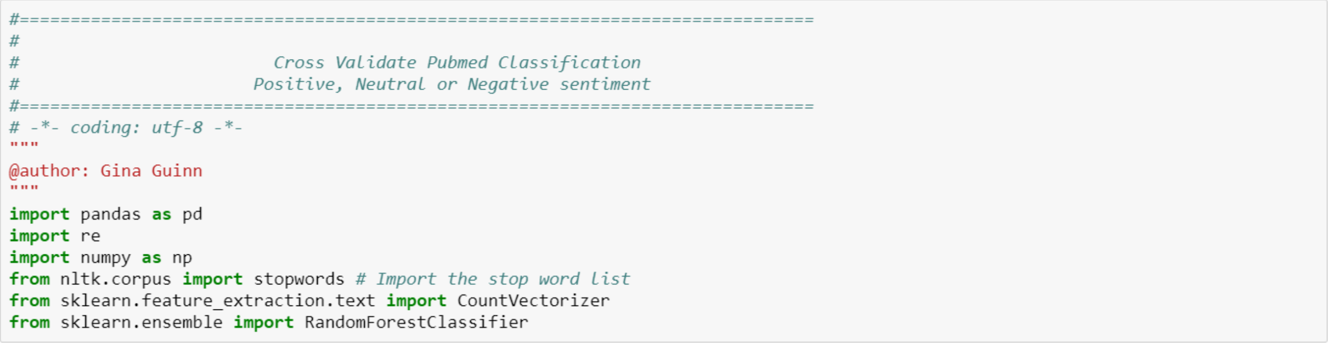
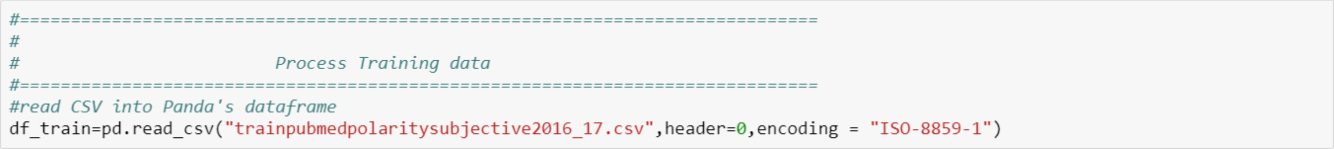
Cross Validation of Pubmed Abstracts using Bag of Words

1) **Import modules**: Using pandas for dataframe processing



```
#==============================================================================
#
#                         Cross Validate Pubmed Classification
#                       Positive, Neutral or Negative sentiment
#==============================================================================
# -*- coding: utf-8 -*-
"""
@author: Gina Guinn
"""
import pandas as pd
import re
import numpy as np
from nltk.corpus import stopwords # Import the stop word list
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestClassifier
```

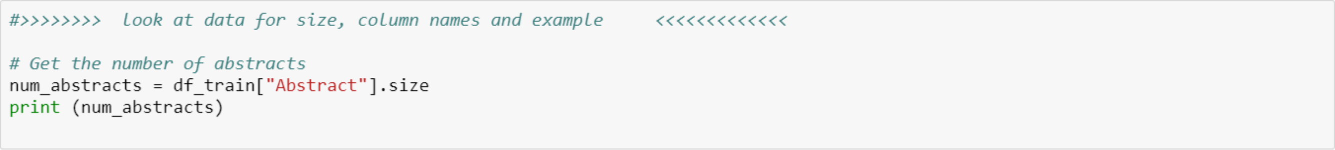
2) **Read CSV:** Unprocessed Training file with Abstracts classified by Pos, Neg, Neutral



```
#==============================================================================
#
#                         Process Training data
#==============================================================================
#read CSV into Panda's dataframe
df_train=pd.read_csv("trainpubmedpolaritysubjective2016_17.csv",header=0,encoding = "ISO-8859-1")
```

3) **File Information:**

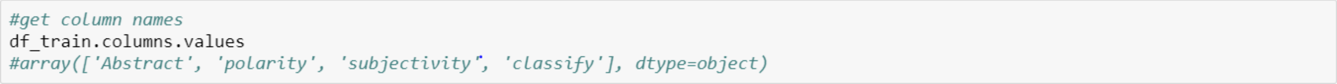
a) Number of records



```
#>>>>>>>>  look at data for size, column names and example     <<<<<<<<<<<<<

# Get the number of abstracts
num_abstracts = df_train["Abstract"].size
print (num_abstracts)
```

```
# Get the number of abstracts
    2966
```

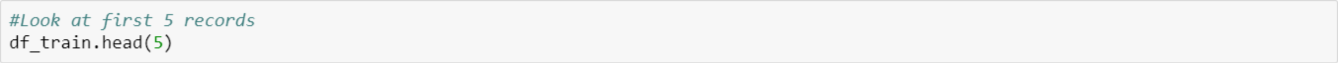
b) Column Names



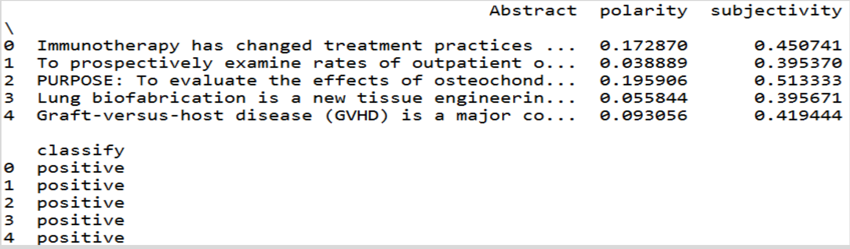
```
#get column names
df_train.columns.values
#array(['Abstract', 'polarity', 'subjectivity', 'classify'], dtype=object)
```

```
#get column names
df_train.columns.values
#array(['Abstract', 'polarity', 'subjectivity', 'classify'], dtype=object)
```

c) Look at first 5 rows



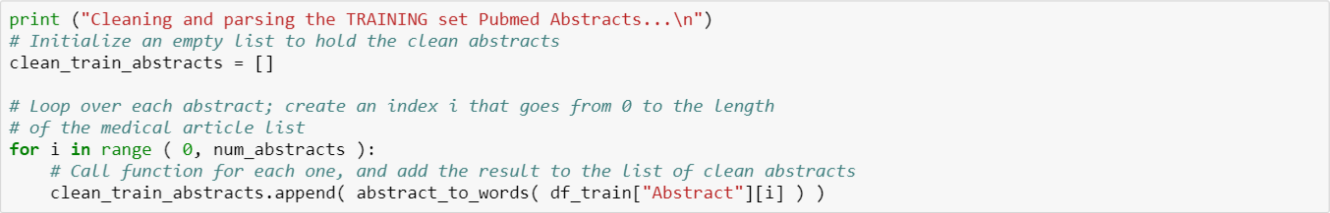
```
#Look at first 5 records
df_train.head(5)
```



```
                                            Abstract  polarity  subjectivity  \
0  Immunotherapy has changed treatment practices ...  0.172870      0.450741
1  To prospectively examine rates of outpatient o...  0.038889      0.395370
2  PURPOSE: To evaluate the effects of osteochond...  0.195906      0.513333
3  Lung biofabrication is a new tissue engineerin...  0.055844      0.395671
4  Graft-versus-host disease (GVHD) is a major co...  0.093056      0.419444

   classify
0  positive
1  positive
2  positive
3  positive
4  positive
```

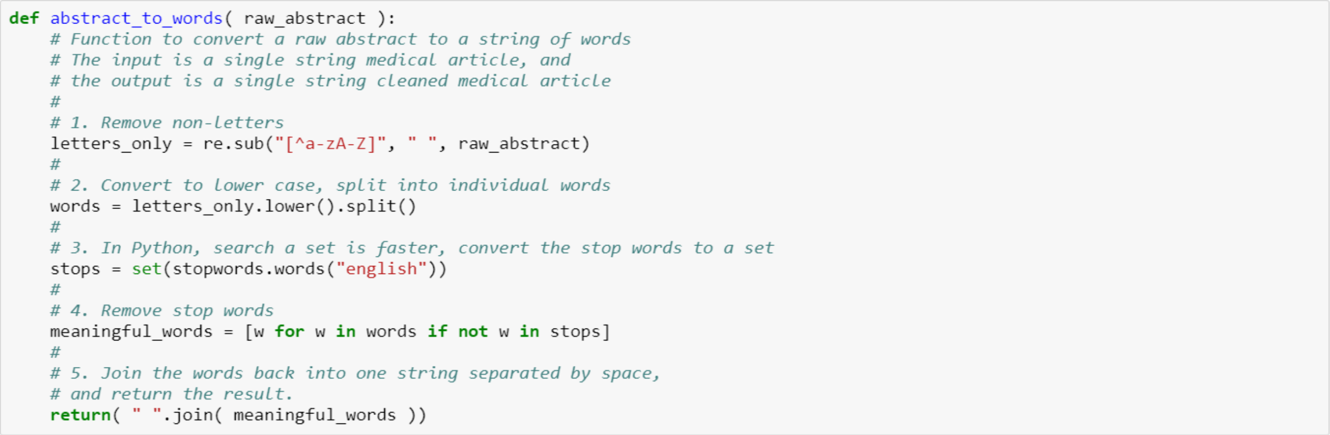
4) **Data Cleaning:**

Process each row

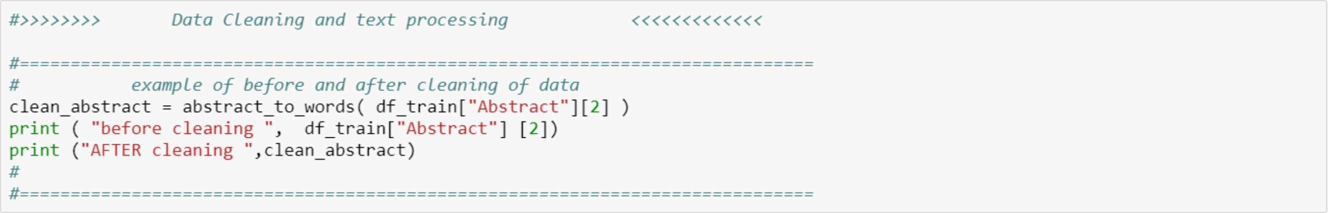


```
print ("Cleaning and parsing the TRAINING set Pubmed Abstracts...\n")
# Initialize an empty list to hold the clean abstracts
clean_train_abstracts = []

# Loop over each abstract; create an index i that goes from 0 to the length
# of the medical article list
for i in range ( 0, num_abstracts ):
    # Call function for each one, and add the result to the list of clean abstracts
    clean_train_abstracts.append( abstract_to_words( df_train["Abstract"][i] ) )
```

1) Remove none letters – use re (regular expression) to remove numbers and punctuation
2) Convert all letters to lower case
3) Search stop words
4) Remove word if found in stopwords



```
def abstract_to_words( raw_abstract ):
    # Function to convert a raw abstract to a string of words
    # The input is a single string medical article, and
    # the output is a single string cleaned medical article
    #
    # 1. Remove non-letters
    letters_only = re.sub("[^a-zA-Z]", " ", raw_abstract)
    #
    # 2. Convert to lower case, split into individual words
    words = letters_only.lower().split()
    #
    # 3. In Python, search a set is faster, convert the stop words to a set
    stops = set(stopwords.words("english"))
    #
    # 4. Remove stop words
    meaningful_words = [w for w in words if not w in stops]
    #
    # 5. Join the words back into one string separated by space,
    # and return the result.
    return( " ".join( meaningful_words ))
```

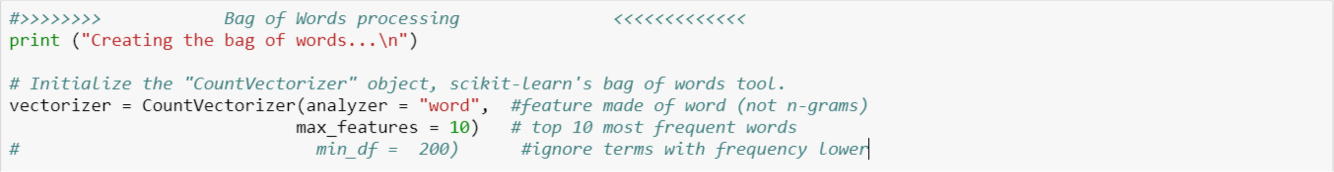


```
#>>>>>>>>>       Data Cleaning and text processing            <<<<<<<<<<<<<

#=============================================================================
#           example of before and after cleaning of data
clean_abstract = abstract_to_words( df_train["Abstract"][2] )
print ( "before cleaning ",  df_train["Abstract"] [2])
print ("AFTER cleaning ",clean_abstract)
#
#=============================================================================
```

```
before cleaning     PURPOSE: To evaluate the effects of osteochondral autograph transplantation (OAT)
mosaicplasty as a concomitant procedure with opening-wedge valgus high tibial osteotomy (HTO) for
spontaneous osteonecrosis of the medial femoral condyle (MFC) on clinical outcomes and cartilage status in
comparison with bone marrow stimulation (BMS) by drilling  and to assess the relation between lesion size
and postoperative cartilage status.METHODS: Fifty-eight patients with spontaneous osteonecrosis of the
MFC were treated with opening-wedge HTO and a concomitant procedure of BMS (28 patients) or OAT (30
patients)…………

AFTER cleaning     purpose evaluate effects osteochondral autograft transplantation oat mosaicplasty
concomitant procedure opening wedge valgus high tibial osteotomy hto spontaneous osteonecrosis medial
femoral condyle mfc clinical outcomes cartilage status comparison bone marrow stimulation bms drilling
assess relation lesion size postoperative cartilage status methods fifty eight patients spontaneous
osteonecrosis mfc treated opening wedge hto concomitant procedure bms patients oat patients
```

5)**Portion of Bag of Words Processing:** use Scikit-learn CountVectorizer

a) max_features use the top frequently used words

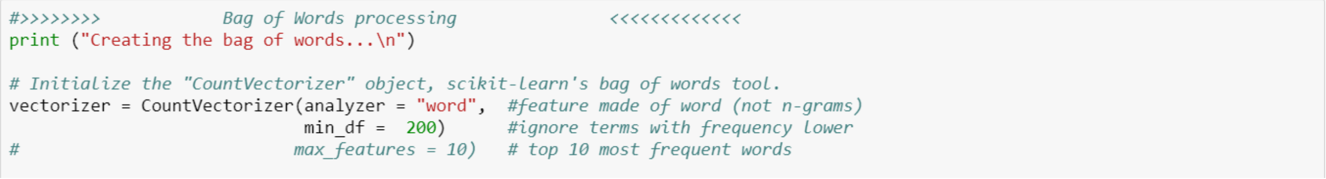


```
#>>>>>>>>            Bag of Words processing               <<<<<<<<<<<<<
print ("Creating the bag of words...\n")

# Initialize the "CountVectorizer" object, scikit-learn's bag of words tool.
vectorizer = CountVectorizer(analyzer = "word",  #feature made of word (not n-grams)
                             max_features = 10)   # top 10 most frequent words
#                              min_df =  200)      #ignore terms with frequency lower
```

```
Creating the bag of words...

(2966, 10)
3589 bone
1924 cd
4691 cell
5448 cells
2087 disease
3231 marrow
6321 patients
2583 stem
3068 transplantation
2086 treatment
```

Bag of Words: Fine tune for Random forest

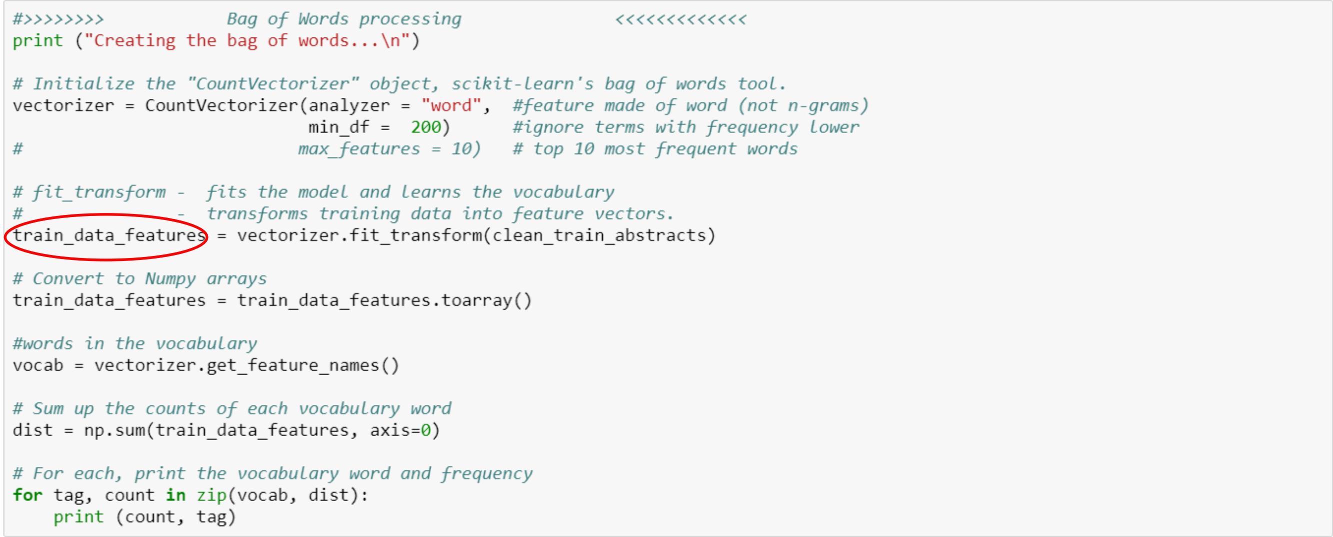
b) min_df ignores word frequency below this threshold



```
#>>>>>>>>            Bag of Words processing               <<<<<<<<<<<<<
print ("Creating the bag of words...\n")

# Initialize the "CountVectorizer" object, scikit-learn's bag of words tool.
vectorizer = CountVectorizer(analyzer = "word",  #feature made of word (not n-grams)
                              min_df =  200)      #ignore terms with frequency lower
#                            max_features = 10)   # top 10 most frequent words
```

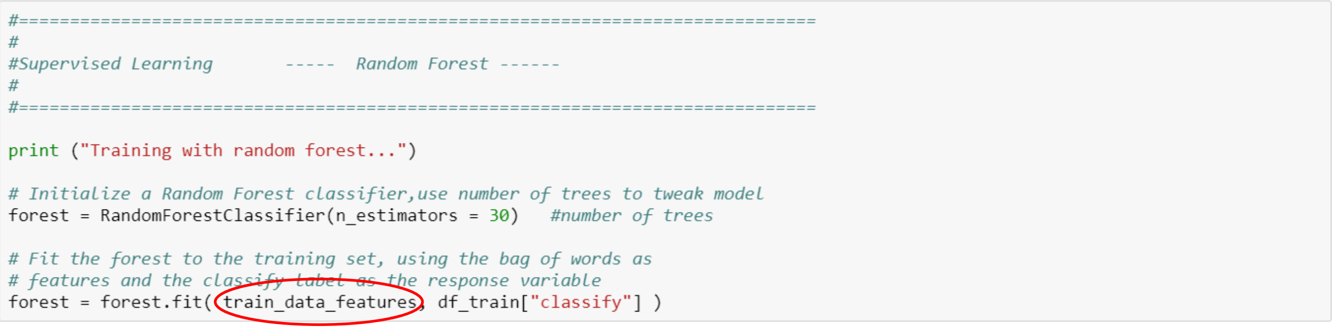
```
Creating the bag of words...

(2966, 252)
324 activation
389 activity
1098 acute
305 addition
361 adult
666 age
271 aim
882 allogeneic
800 also
379 although
379 among……..
```

6)**Full Bag of Words Processing:**



```
#>>>>>>>>            Bag of Words processing               <<<<<<<<<<<<<
print ("Creating the bag of words...\n")

# Initialize the "CountVectorizer" object, scikit-learn's bag of words tool.
vectorizer = CountVectorizer(analyzer = "word",  #feature made of word (not n-grams)
                             min_df =  200)      #ignore terms with frequency lower
#                           max_features = 10)   # top 10 most frequent words

# fit_transform -  fits the model and learns the vocabulary
#               -  transforms training data into feature vectors.
train_data_features = vectorizer.fit_transform(clean_train_abstracts)

# Convert to Numpy arrays
train_data_features = train_data_features.toarray()

#words in the vocabulary
vocab = vectorizer.get_feature_names()

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

# For each, print the vocabulary word and frequency
for tag, count in zip(vocab, dist):
    print (count, tag)
```

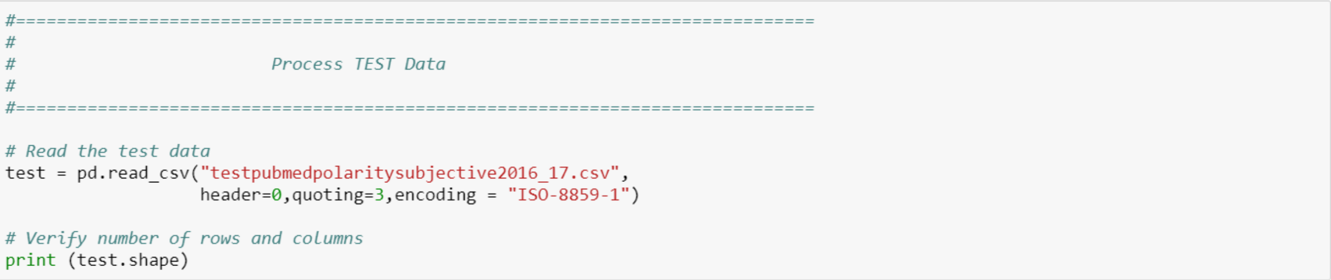
7)**Random Forest:** use Scikit-learn RandomForestClassifier. Fits a number of decision tree classifiers on sub-samples and averages response. Use bag of words to determine how to classify each abstract as Positive, Negative or Neutral in sentiment.



```
#==============================================================================
#
#Supervised Learning       -----  Random Forest ------
#
#==============================================================================

print ("Training with random forest...")

# Initialize a Random Forest classifier,use number of trees to tweak model
forest = RandomForestClassifier(n_estimators = 30)   #number of trees

# Fit the forest to the training set, using the bag of words as
# features and the classify label as the response variable
forest = forest.fit( train_data_features, df_train["classify"] )
```

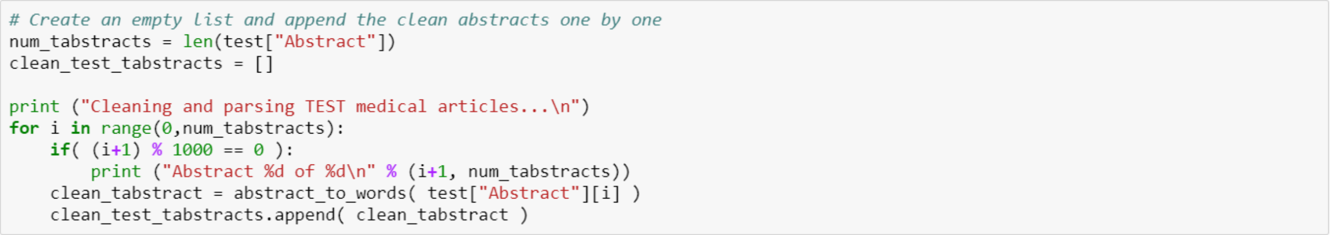
COMPLETED TRAINING

7)**Process Test data for Cross Validation**



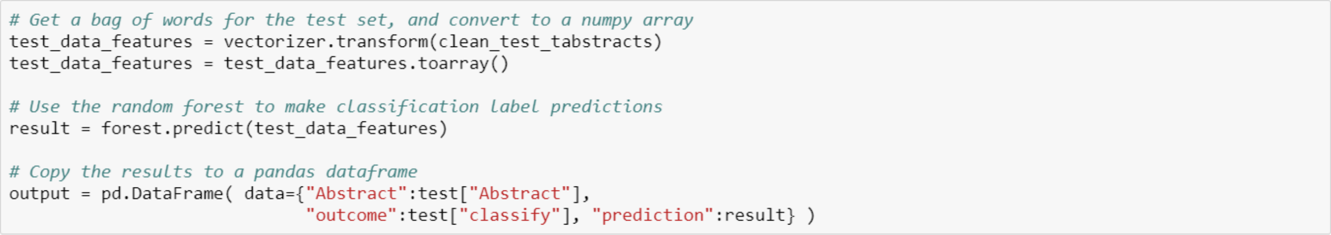
```
#==============================================================================
#
#                         Process TEST Data
#
#==============================================================================

# Read the test data
test = pd.read_csv("testpubmedpolaritysubjective2016_17.csv",
                   header=0,quoting=3,encoding = "ISO-8859-1")

# Verify number of rows and columns
print (test.shape)
```

8)**Clean Test Data:** same process used for training data

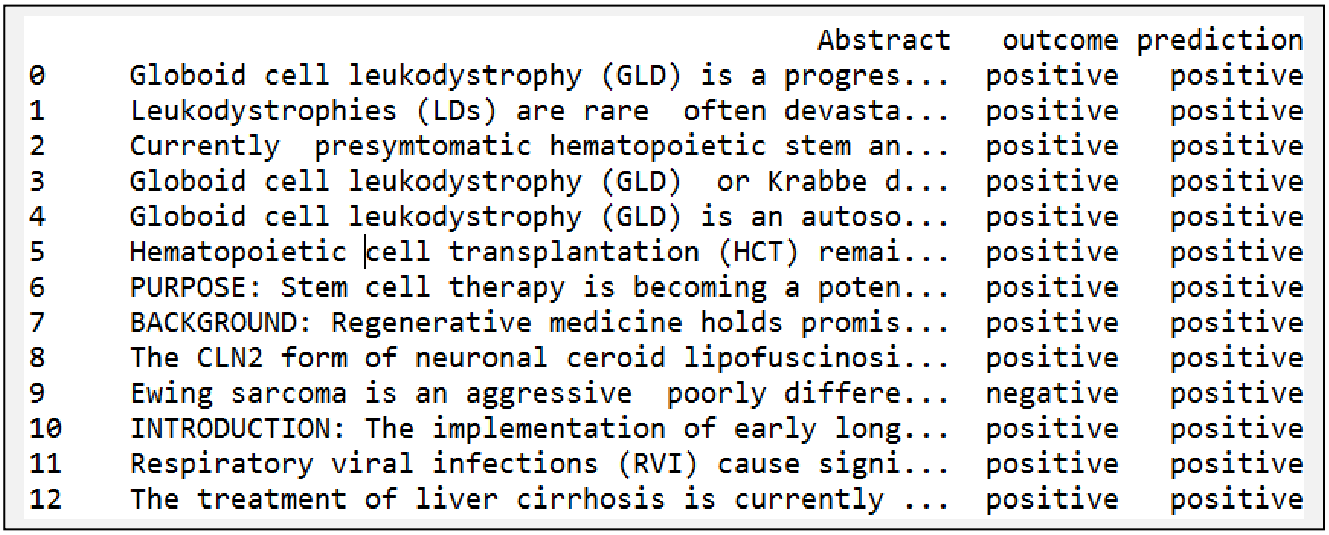


```
# Create an empty list and append the clean abstracts one by one
num_tabstracts = len(test["Abstract"])
clean_test_tabstracts = []

print ("Cleaning and parsing TEST medical articles...\n")
for i in range(0,num_tabstracts):
    if( (i+1) % 1000 == 0 ):
        print ("Abstract %d of %d\n" % (i+1, num_tabstracts))
    clean_tabstract = abstract_to_words( test["Abstract"][i] )
    clean_test_tabstracts.append( clean_tabstract )
```

8)**Bag of Words processing and Random Forest**

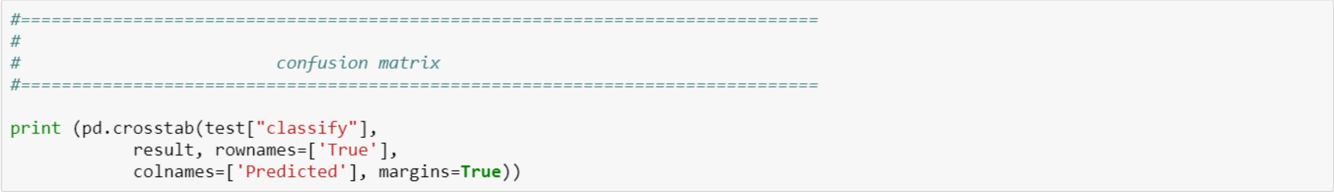


```
# Get a bag of words for the test set, and convert to a numpy array
test_data_features = vectorizer.transform(clean_test_tabstracts)
test_data_features = test_data_features.toarray()

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

# Copy the results to a pandas dataframe
output = pd.DataFrame( data={"Abstract":test["Abstract"],
                             "outcome":test["classify"], "prediction":result} )
```

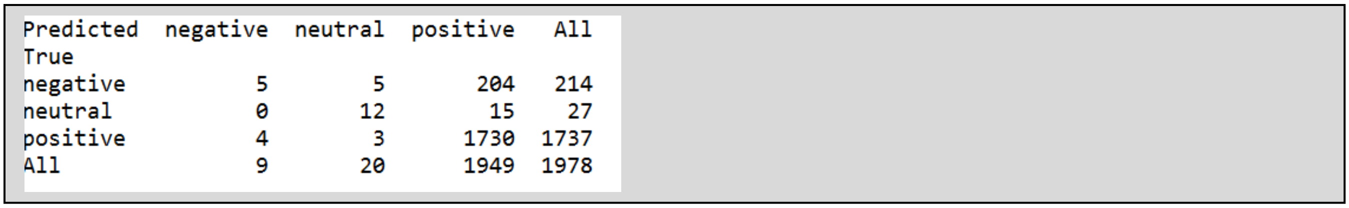


```
                                             Abstract   outcome prediction
0     Globoid cell leukodystrophy (GLD) is a progres... positive   positive
1     Leukodystrophies (LDs) are rare  often devasta... positive   positive
2     Currently  presymtomatic hematopoietic stem an... positive   positive
3     Globoid cell leukodystrophy (GLD)  or Krabbe d... positive   positive
4     Globoid cell leukodystrophy (GLD) is an autoso... positive   positive
5     Hematopoietic cell transplantation (HCT) remai... positive   positive
6     PURPOSE: Stem cell therapy is becoming a poten... positive   positive
7     BACKGROUND: Regenerative medicine holds promis... positive   positive
8     The CLN2 form of neuronal ceroid lipofuscinosi... positive   positive
9     Ewing sarcoma is an aggressive  poorly differe... negative   positive
10    INTRODUCTION: The implementation of early long... positive   positive
11    Respiratory viral infections (RVI) cause signi... positive   positive
12    The treatment of liver cirrhosis is currently ... positive   positive
```

9)**Assess outcome:** Confusion Matrix

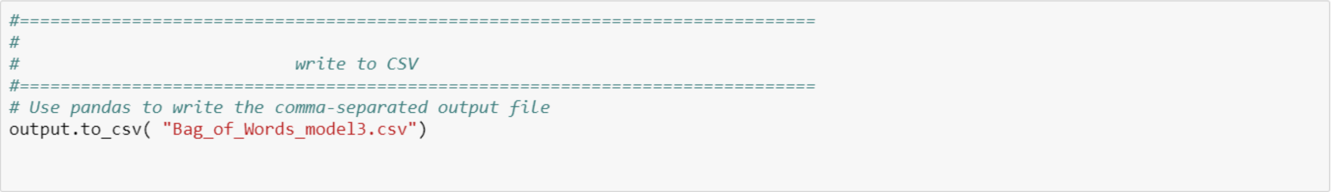


```
#==============================================================================
#
#                          confusion matrix
#==============================================================================

print (pd.crosstab(test["classify"],
            result, rownames=['True'],
            colnames=['Predicted'], margins=True))
```



```
Predicted  negative  neutral  positive   All
True
negative          5        5       204   214
neutral           0       12        15    27
positive          4        3      1730  1737
All               9       20      1949  1978
```

10)**Write to CSV:**



```
#==============================================================================
#
#                            write to CSV
#==============================================================================
# Use pandas to write the comma-separated output file
output.to_csv( "Bag_of_Words_model3.csv")
```