```
import nltk
from nltk.tokenize import word_tokenize
import csv
import string
from nltk.tokenize import RegexpTokenizer


def search(searcharray, inputstring):  # Finds the author and all their books
    flag = False
    filname = 'Profile list'
    with open(filname) as file:
        fl = file.readlines()  # read all file contents into fl
        file.close

    for x in fl:  # for each entry in the list
        temp = x
        if temp[0] == '$' and temp[1] == "A":
            flag = False
            temp = temp[2:-1]  # Cut the sign
            for y in inputstring:  # looking for all entries with the same author
                if temp == y:  # if the selected author is the one sought after
                    flag = True
                    name = temp
        if flag == True and temp != y:
            booknamearr.append(temp)  # Assign the book name
            temp = name + '_' + temp
            autharray.append(name)  # assign the authorname
            searcharray.append(temp)  # Assigns the full path
    return searcharray


def loadina(path, firstpart, find):  # This one loads in the selected books by author
    filist = []
    internalpath = path + '/'
    count = 0
    for y in find:
        new = firstpart + y
        new = new.replace('\n', '')
        new = internalpath + new
        print(new)
        try:
            with open(new) as file:  # Opens the filename
                fl = file.readlines()
                filist.append(fl)
                fl = ''
                file.close
            pass
        except:  # If it doesn't exist because e.g: $ is used then it skips that
            print(new, 'Does not exist in the filepath!')
            autharray.pop(count)  # likewise removes the same entry from the author and boo
k title arrays
            booknamearr.pop(count)
            continue  # carry on to the next entry
        count += 1

    return filist


def number_of_sentences(doc):  # includes punctuation
    resultlist = []
    for x in doc:
        resultlist.append(len(x))
    return resultlist


def calculatemean(autharray, resultlist):  # My test program calculates the mean, you can p
aste your progs here
    count = 1
    mean = 0
    interarray = []
    takearray = autharray
    takearray.append('stopper')  # For mine to work and process the last value needed to be
 appended
    for x in range(0, len(takearray) - 1):
        if takearray[x] == takearray[x + 1]:
            mean = mean + resultlist[x]
            count += 1
        else:
            mean = mean + resultlist[x]
            mean = mean / count
            interarray.append(autharray[x])
            interarray.append(mean)
            mean = 0
            count = 1
    autharray.remove(autharray[-1])
    return interarray


def concat_results(autharray, resultlist):
    finalarray = []
    for i in range(len(resultlist)):
        finalarray.append(autharray[i])
        finalarray.append(resultlist[i])
    return finalarray


autharray = []  # Takes the name of each author
booknamearr = []  # Takes the name of the book


def counting_vocabulary(doc):
    """Counting vocabulary of a text"""
    resultlist = []
    for x in doc:
        string = ''
        for i in x:
            string += i
        tokens_text = word_tokenize(string)
        resultlist.append(len(tokens_text))
    return resultlist


def lexical_richness(doc):
    """Counting lexical richness"""
    resultlist = []
    for x in doc:
        string = ''
        for i in x:
            string += i
        tokens_text = word_tokenize(string)
        vocabulary = len(set(tokens_text))
        resultlist.append((vocabulary / len(tokens_text)) * 100)
    return resultlist


def number_of_punctuation(doc):  # number of punctuation signs in a list of books
    resultlist = []
    for x in doc:
        text = ''
        for i in x:
            text += i
        punctuation = string.punctuation
        n = 0
        for l in text:
            if l in punctuation:
                n += 1
        resultlist.append(n)
    return resultlist


def number_of_words_np(doc):  # no punctuation
    resultlist = []
    for x in doc:
        string = ''
        for i in x:
            string += i
        tokenizer = RegexpTokenizer(r'\w+')
        num = len(tokenizer.tokenize(string))
        resultlist.append(num)
    return resultlist


def words_per_sentence(doc):  # average number of words per sentence (includes puncutation)
    l1 = counting_vocabulary(doc)
    l2 = number_of_sentences(doc)
    num = [x / y for x, y in zip(l1, l2)]
    return num


def words_per_sentence_np(doc):  # average number of words per sentence (no punctuation)
    l1 = number_of_words_np(doc)
    l2 = number_of_sentences(doc)
    num = [x / y for x, y in zip(l1, l2)]
    return num


def punctuation_per_sentence(doc):  # number of punct per sentences
    l1 = number_of_punctuation(doc)
    l2 = number_of_sentences(doc)
    num = [x / y for x, y in zip(l1, l2)]
    return num


def number_of_letters(doc):  # includes punctuation
    resultlist = []
    for x in doc:
        string = ''
        for i in x:
            string += i
        num_l = len(string)
        resultlist.append(num_l)
    return resultlist


def number_of_letters_np(doc):  # no punctuation
    l1 = number_of_letters(doc)
    l2 = number_of_punctuation(doc)
    num = [x - y for x, y in zip(l1, l2)]
    return num


def average_word_length_np(doc):  # average length words per doc
    l1 = number_of_letters_np(doc)
    l2 = number_of_words_np(doc)
    num = [x / y for x, y in zip(l1, l2)]
    return num


def egocentricity_level(doc):
    """Return frequency value of egocentric words for a list of books"""
    ego_list = ["me", "Me", "myself", "Myself", "I", "mine", "Mine", "my", "My"]
    result_l = []
    l2 = number_of_words_np(doc)
    for x in doc:
        n = 0
        string = ''
        for i in x:
            string += i
        for w in word_tokenize(string):
            for item in ego_list:
                if w == item:
                    n += 1
        result_l.append(n)
    num = [x / y for x, y in zip(result_l, l2)]
    return num


def modal_verbs_frequency(doc):
    """Return frequency of modal verbs for a list of books"""
    modal_list = ["can", "could", "could have", "must", "need", "must have", "may", "might"
, "would", "would have",
                  "shall", "need", "have to", "ought to", "dare", "should", "should have",
"will be able to", "forced",
                  "will have to", "allowed", "Can", "Could", "Could have", "Must", "Need",
"Must have", "May", "Might",
                  "Would", "Would have", "Shall", "Need", "Have to", "Ought to", "Dare", "S
hould", "Should have",
                  "Will be able to", "Forced", "Will have to", "Allowed"]
    result_l = []
    l2 = number_of_words_np(doc)
    for x in doc:
        string = ''
        n = 0
        for i in x:
            string += i
        for w in word_tokenize(string):
            for item in modal_list:
                if w == item:
                    n += 1
        result_l.append(n)
    num = [x / y for x, y in zip(result_l, l2)]
    return num


def connector_frequency(doc):
    """ Return frequency of different type of connectors for a list of books"""
    emphasis_list = ["Especially", "Also", "In particular", "Furthermore", "In addition", "
Indeed", "Of course",
                     "Certainly", "Above all", "Specifically", "Significantly", "Notably"]
    emphasis_list2 = []
    comparison_list = ["As if", "As", "Equally", "Similarly", "In the same way", "Comparabl
e", "In like manner",
                       "Alternatively", "Unless", "Despite this", "By the way"]
    comparison_list2 = []
    contrast_list = ["But", "However", "On the other hand", "Otherwise", "Unlike", "Convers
ely", "At the same time",
                     "In spite of", "Whereas", "While", "Yet", "Apart from"]
    contrast_list2 = []
    addition_list = ["As well as", "Further", "Furthermore", "And then", "And", "Too", "Als
o", "In addition to",
                     "Not only", "Or"]
    addition_list2 = []
    illustration_list = ["Such as", "In this case", "For one thing", "For instance", "For e
xample", "In the case of",
                         "Illustrated by", "As an example", "As instance", "In other words"
]
    illustration_list2 = []
    for item in emphasis_list:
        emphasis_list2.append(item.lower())
    emphasis_list.extend(comparison_list2)
    for item in comparison_list:
        comparison_list2.append(item.lower())
    comparison_list.extend(comparison_list2)
    for item in contrast_list:
        contrast_list2.append(item.lower())
    contrast_list.extend(contrast_list2)
    for item in addition_list:
        addition_list2.append(item.lower())
    addition_list.extend(addition_list2)
    for item in illustration_list:
        illustration_list2.append(item.lower())
    illustration_list.extend(illustration_list2)
    final1 = []
    final2 = []
    final3 = []
    final4 = []
    final5 = []
    l2 = number_of_words_np(doc)
    for x in doc:
        l = []
        for i in range(5):
            l.append(l2[0])
        result_l = []
        string = ’’
        emphasis = comparison = contrast = addition = illustration = 0
        for i in x:
            string += i
        for w in word_tokenize(string):
            if w in emphasis_list:
                emphasis += 1
            if w in comparison_list:
                comparison += 1
            if w in contrast_list:
                contrast += 1
            if w in addition_list:
                addition += 1
            if w in illustration_list:
                illustration += 1
        result_l.append(emphasis)
        result_l.append(comparison)
        result_l.append(contrast)
        result_l.append(addition)
        result_l.append(illustration)
        num = [x / y for x, y in zip(result_l, l)]
        final1.append(num[0])
        final2.append(num[1])
        final3.append(num[2])
        final4.append(num[3])
        final5.append(num[4])
        l2.pop(0)
    return final1, final2, final3, final4, final5


def type_of_punctuation(
        doc):
    """Return a list of frequencies of different punctuations of a list of books: comma, do
tcomma, exclamation, question mark, ..."""
    final1 = []
    final2 = []
    final3 = []
    final4 = []
    final5 = []
    final6 = []
    final7 = []
    l2 = number_of_punctuation(doc)
    for x in doc:
        comma = dotcomma = excl = qmark = dash = slash = twodot = 0
        n = []
        for i in range(7):
            n.append(l2[0])
        result_l = []
        string = ''
        for i in x:
            string += i
        for l in string:
            if l == ',':
                comma += 1
            elif l == ';':
                dotcomma += 1
            elif l == '!':
                excl += 1
            elif l == '?':
                qmark += 1
            elif l == '-':
                dash += 1
            elif l == '/':
                slash += 1
            elif l == ':':
                twodot += 1
        result_l.append(comma)
        result_l.append(dotcomma)
        result_l.append(excl)
        result_l.append(qmark)
        result_l.append(dash)
        result_l.append(slash)
        result_l.append(twodot)
        num = [x / y for x, y in zip(result_l, n)]
        final1.append(num[0])
        final2.append(num[1])
        final3.append(num[2])
        final4.append(num[3])
        final5.append(num[4])
        final6.append(num[5])
        final7.append(num[6])
        l2.pop(0)
    return final1, final2, final3, final4, final5, final6, final7


def pos_tag_frequency(doc):
    """Return frequency of each POS tag of a list of books"""
    list_pos = ["CC", "CD", "DT", "EX", "FW", "IN", "JJ", "JJR", "JJS", "LS", "MD", "NN", "
NNS", "NNP", "NNPS",
                "PDT",
                "POS", "PRP", "PRP$", "RB", "RBR", "RBS", "RP", "TO", "UH", "VB", "VBD", "V
BG", "VBN", "VBP",
                "VBZ",
                "WDT", "WP", "WP$", "WRB"]
    obj = {}
    for i in range(35):
        obj[str(i)] = []
    l2 = number_of_words_np(doc)
    for x in doc:
        l = []
        for i in range(35):
            l.append(l2[0])
        string = ''
        for i in x:
            string += i
        list_pos_numb = []
        for i in range(0, len(list_pos)):
            list_pos_numb.append(0)
        text = nltk.pos_tag(word_tokenize(string))
        for elem in text:
            for pos in list_pos:
                if elem[1] == pos:
                    list_pos_numb[list_pos.index(elem[1])] += 1
        num = [x / y for x, y in zip(list_pos_numb, l)]
        for m in range(len(num)):
            obj[str(m)].append(num[m])
    return obj


def loadinbooks():  # loads in the selected books/authors
    path = '/home/macaire/Bureau/test'  # specify the path
    inputlist = ['Addams', 'Bacon', 'Bentham', 'Berkeley', 'Burke', 'Cavendish', 'Clifford'
, 'Dewey',
                 'Emerson', 'Fellerton', 'Godwin', 'Goldman', 'Hide', 'Hobbes',
                 'Hume', 'James', 'Jordan', 'Locke', 'Martineau', 'Mill',
                 'Russell', 'Santayana', 'Sidgwick', 'Smith', 'Spencer', 'Thoreau',
                 'Wollstonecraft', 'unknow']  # You select which authors to test on here!
    find = []
    search(find, inputlist)  # Assign the filename as well as initialise author and booknam
e
    firstpart = 'processedbysentences_'  # specify what type of file do you want to test on
    bookid = loadina(path, firstpart, find)  # load the contents of the specified filename
list
    return bookid, find


bookid, find = loadinbooks()


def displayscore(resultlist, authorsummary):
    for x in range(0, len(resultlist)):
        print(autharray[x], booknamearr[x], resultlist[x])
    print(authorsummary)


def create_csv(filename, *args):
    with open(filename, 'w') as csvfile:
        filewriter = csv.writer(csvfile, delimiter=',')
        filewriter.writerow(['Authors', 'av_sentences', 'av_voc', 'lex_richness', 'av_punct
', 'av_num_word_ns',
                             'av_num_words_sentences', 'av_num_words_sentences_ns', 'av_num
_ns_sentences',
                             'av_words_length', 'av_ego', 'av_modal', 'emphasis', 'comparis
on', 'contrast', 'addition',
                             'illustration', 'comma', 'dotcomma', 'excl', 'qmark', 'dash',
'slash', 'twodot', "CC",
                             "CD", "DT", "EX", "FW", "IN", "JJ", "JJR", "JJS", "LS", "MD",
"NN", "NNS", "NNP", "NNPS",
                             "PDT",
                             "POS", "PRP", "PRP$", "RB", "RBR", "RBS", "RP", "TO", "UH", "V
B", "VBD", "VBG", "VBN",
                             "VBP", "VBZ",
                             "WDT", "WP", "WP$", "WRB"])
        i = 0
        while i < len(args[0]):
            res = []
            j = 0
            for l in args:
                if j == 0:
                    res.append(l[i])
                    res.append(l[i + 1])
                else:
                    res.append(l[i + 1])
                j += 1
            filewriter.writerow(res)
            i += 2


resultlist = number_of_sentences(bookid)  # find the total len of sentences of the books
# For your programs, here you need to populate the result list
# with your results, then print out the score by passing it further
resultlist2 = counting_vocabulary(bookid)  # find the vocabulary size per books
resultlist3 = lexical_richness(bookid)  # find the percentage of lexical richness per books
```

```python
resultlist4 = number_of_punctuation(bookid)  # find the number of punctuations signs per books
resultlist5 = number_of_words_np(bookid)  # find the number of words without punctuation signs per books
resultlist6 = words_per_sentence(bookid)  # find the number of words per sentences per books
resultlist7 = words_per_sentence_np(bookid)  # find the number of words per sentences with no punctuation per books
resultlist8 = punctuation_per_sentence(bookid)  # find the number of punctuation signs per sentence per books
resultlist9 = average_word_length_np(bookid)  # find the words length per books
resultlist10 = egocentricity_level(bookid)  # find the frequence of egocentric words per books
resultlist11 = modal_verbs_frequency(bookid)  # find frequence of modal words per books
r1, r2, r3, r4, r5 = connector_frequency(bookid)  # find percentage of connector words per books
p1, p2, p3, p4, p5, p6, p7 = type_of_punctuation(bookid)  # find percentage of type of punctuation per books
pos = pos_tag_frequency(bookid)  # find percentage of POS tag per books

# compute average between books which belongs to the same author
c1 = calculatemean(autharray, resultlist)  # average len of sentences
c2 = calculatemean(autharray, resultlist2)  # average vocabulary size
c3 = calculatemean(autharray, resultlist3)  # average percentage of lexical richness
c4 = calculatemean(autharray, resultlist4)  # average number of punctuations signs
c5 = calculatemean(autharray, resultlist5)  # average number of words without punctuation signs
c6 = calculatemean(autharray, resultlist6)  # average number of words per sentences
c7 = calculatemean(autharray, resultlist7)  # average number of words per sentences with no punctuation
c8 = calculatemean(autharray, resultlist8)  # average number of punctuation signs per sentence
c9 = calculatemean(autharray, resultlist9)  # average words length
c10 = calculatemean(autharray, resultlist10)  # average frequence of egocentric words
c11 = calculatemean(autharray, resultlist11)  # average frequence of modal words
con1 = calculatemean(autharray, r1)  # average frequence of emphasis connector words
con2 = calculatemean(autharray, r2)  # average frequence of comparison connector words
con3 = calculatemean(autharray, r3)  # average frequence of contrast connector words
con4 = calculatemean(autharray, r4)  # average frequence of addition connector words
con5 = calculatemean(autharray, r5)  # average frequence of illustration connector words
punct1 = calculatemean(autharray, p1)  # average frequence of comma punctuations
punct2 = calculatemean(autharray, p2)  # average frequence of dotcomma punctuations
punct3 = calculatemean(autharray, p3)  # average frequence of excl punctuations
punct4 = calculatemean(autharray, p4)  # average frequence of qmark punctuations
punct5 = calculatemean(autharray, p5)  # average frequence of dash punctuations
punct6 = calculatemean(autharray, p6)  # average frequence of slash punctuations
punct7 = calculatemean(autharray, p7)  # average frequence of twodot punctuations
pos_tag = {}
for i in range(35):
    pos_tag[str(i)] = []
for key, val in pos.items():
    pos_tag[key] = calculatemean(autharray, val)  # average frequence of each POS tags

# create csv file with all the stat values previously computed
create_csv('data_authors2.csv', c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, con1, con2, con3, con4, con5, punct1,
           punct2, punct3, punct4,
           punct5, punct6, punct7, pos_tag['0'], pos_tag['1'], pos_tag['2'], pos_tag['3'], pos_tag['4'], pos_tag['5'],
           pos_tag['6'], pos_tag['7'],
           pos_tag['8'], pos_tag['9'], pos_tag['10'], pos_tag['11'], pos_tag['12'], pos_tag['13'], pos_tag['14'],
           pos_tag['15'], pos_tag['16'], pos_tag['17'],
           pos_tag['18'], pos_tag['19'], pos_tag['20'], pos_tag['21'], pos_tag['22'], pos_tag['23'], pos_tag['24'],
           pos_tag['25'], pos_tag['26'], pos_tag['27'],
           pos_tag['28'], pos_tag['29'], pos_tag['30'], pos_tag['31'], pos_tag['32'], pos_tag['33'], pos_tag['34'])
```

```
# displayscore(resultlist2, c1)  # Pass the result list for individual books and for profil
e
```