Assignment 2

October 2, 2020

---

*You are currently looking at **version 1.0** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

---

1 Assignment 2 - Introduction to NLTK

In part 1 of this assignment you will use nltk to explore the Herman Melville novel Moby Dick. Then in part 2 you will create a spelling recommender function that uses nltk to find words similar to the misspelling.

1.1 Part 1 - Analyzing Moby Dick

```
In [4]: import nltk
        import pandas as pd
        import numpy as np

        #nltk.download('punkt')

        #nltk.download('gutenberg')

        #nltk.download('genesis')

        #nltk.download('inaugural')

        #nltk.download('nps_chat')

        #nltk.download('webtext')

        #nltk.download('treebank')

        #nltk.download('udhr')

        #nltk.download('tagsets')
        #nltk.download('averaged_perceptron_tagger')

        #nltk.download('words')

        # If you would like to work with the raw text you can use 'moby_raw'
        with open('moby.txt', 'r') as f:
            moby_raw = f.read()

        # If you would like to work with the novel in nltk.Text format you can use 'text1'
        moby_tokens = nltk.word_tokenize(moby_raw)
        text1 = nltk.Text(moby_tokens)
```

```
[nltk_data] Downloading package punkt to /home/jovyan/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package gutenberg to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/gutenberg.zip.
[nltk_data] Downloading package genesis to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/genesis.zip.
[nltk_data] Downloading package inaugural to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/inaugural.zip.
[nltk_data] Downloading package nps_chat to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/nps_chat.zip.
[nltk_data] Downloading package webtext to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/webtext.zip.
[nltk_data] Downloading package treebank to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
[nltk_data] Downloading package udhr to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/udhr.zip.
[nltk_data] Downloading package tagsets to /home/jovyan/nltk_data...
[nltk_data]   Unzipping help/tagsets.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/jovyan/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package words to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
```

1.1.1 Example 1

How many tokens (words and punctuation symbols) are in text1?

This function should return an integer.

```
In [2]: def example_one():

            return len(nltk.word_tokenize(moby_raw)) # or alternatively len(text1)

        example_one()
```

```
Out[2]: 254989
```

1.1.2 Example 2

How many unique tokens (unique words and punctuation) does text1 have?

This function should return an integer.

```
In [3]: def example_two():

            return len(set(nltk.word_tokenize(moby_raw))) # or alternatively len(set(text1))

        example_two()
```

```
Out[3]: 20755
```

1.1.3 Example 3

After lemmatizing the verbs, how many unique tokens does text1 have?

This function should return an integer.

```
In [4]: from nltk.stem import WordNetLemmatizer

        def example_three():

            lemmatizer = WordNetLemmatizer()
            lemmatized = [lemmatizer.lemmatize(w,'v') for w in text1]

            return len(set(lemmatized))

        example_three()
```

```
Out[4]: 16900
```

1.1.4 Question 1

What is the lexical diversity of the given text input? (i.e. ratio of unique tokens to the total number of tokens)

This function should return a float.

```
In [5]: def answer_one():

            return example_two()/example_one()

        answer_one()
```

```
Out[5]: 0.08139566804842562
```

1.1.5 Question 2

What percentage of tokens is ’whale’or ’Whale’?

This function should return a float.

```
In [6]: def answer_two():
            from nltk.probability import FreqDist

            f1 = FreqDist(text1)

            return (f1['whale']+f1['Whale'])/example_one()*100

        answer_two()

Out[6]: 0.4125668166077752
```

1.1.6 Question 3

What are the 20 most frequently occurring (unique) tokens in the text? What is their frequency?

This function should return a list of 20 tuples where each tuple is of the form `(token, frequency)`. The list should be sorted in descending order of frequency.

```
In [7]: def answer_three():

            from operator import itemgetter
            from nltk.probability import FreqDist

            f2 = FreqDist(text1)

            return sorted(f2.items(), key=itemgetter(1), reverse=True)[:20]

        answer_three()

Out[7]: [(',', 19204),
         ('the', 13715),
         ('.', 7308),
         ('of', 6513),
         ('and', 6010),
         ('a', 4545),
         ('to', 4515),
         (';', 4173),
         ('in', 3908),
         ('that', 2978),
         ('his', 2459),
         ('it', 2196),
         ('I', 2097),
         ('!', 1767),
         ('is', 1722),
         ('--', 1713),
         ('with', 1659),
         ('he', 1658),
         ('was', 1639),
         ('as', 1620)]
```

1.1.7 Question 4

What tokens have a length of greater than 5 and frequency of more than 150?

This function should return an alphabetically sorted list of the tokens that match the above constraints. To sort your list, use `sorted()`

```
In [8]: def answer_four():
            from nltk.probability import FreqDist

            f3 = FreqDist(text1)

            return sorted([w for w in f3.keys() if len(w)>5 and f3[w]>150])

        answer_four()
```

```
Out[8]: ['Captain',
         'Pequod',
         'Queequeg',
         'Starbuck',
         'almost',
         'before',
         'himself',
         'little',
         'seemed',
         'should',
         'though',
         'through',
         'whales',
         'without']
```

1.1.8 Question 5

Find the longest word in text1 and that word's length.

This function should return a tuple `(longest_word, length)`.

```
In [9]: def answer_five():

            longest = (0,0)

            for w in text1:
                if len(w) > longest[1]:
                    longest = (w,len(w))
                else:
                    continue

            return longest

        answer_five()
```

```
Out[9]: ("twelve-o'clock-at-night", 23)
```

1.1.9 Question 6

What unique words have a frequency of more than 2000? What is their frequency?

"Hint: you may want to use `isalpha()` to check if the token is a word and not punctuation."

This function should return a list of tuples of the form `(frequency, word)` sorted in descending order of frequency.

```
In [10]: def answer_six():

             from nltk.probability import FreqDist

             f4 = FreqDist(text1)

             return sorted([(f4[w],w) for w in f4 if w.isalpha()==True and f4[w]>2000], key = la

         answer_six()
```

```
Out[10]: [(13715, 'the'),
          (6513, 'of'),
          (6010, 'and'),
          (4545, 'a'),
          (4515, 'to'),
          (3908, 'in'),
          (2978, 'that'),
          (2459, 'his'),
          (2196, 'it'),
          (2097, 'I')]
```

1.1.10 Question 7

What is the average number of tokens per sentence?

This function should return a float.

```
In [11]: def answer_seven():

             sentences = nltk.sent_tokenize(moby_raw)

             l2=list()
             for sentence in sentences:
                 words = nltk.word_tokenize(sentence)
                 l2.append(len(words))

             return sum(l2)/len(l2)

         answer_seven()
```

```
Out[11]: 25.881952902963864
```

1.1.11 Question 8

What are the 5 most frequent parts of speech in this text? What is their frequency?

This function should return a list of tuples of the form `(part_of_speech, frequency)` *sorted in descending order of frequency.*

```
In [18]: def answer_eight():

             from collections import Counter

             pos_tags = nltk.pos_tag(text1)

             tags = [x[1] for x in pos_tags]

             counts = Counter(tags)

             return sorted(list(zip(counts.keys(),counts.values())), key = lambda x: x[1], rever

         answer_eight()
```

```
Out[18]: [('NN', 32730), ('IN', 28657), ('DT', 25867), (',', 19204), ('JJ', 17620)]
```

1.2 Part 2 - Spelling Recommender

For this part of the assignment you will create three different spelling recommenders, that each take a list of misspelled words and recommends a correctly spelled word for every word in the list.

For every misspelled word, the recommender should find find the word in `correct_spellings` that has the shortest distance*, and starts with the same letter as the misspelled word, and return that word as a recommendation.

*Each of the three different recommenders will use a different distance measure (outlined below).

Each of the recommenders should provide recommendations for the three default words provided: `['cormulent', 'incendenece', 'validrate']`.

```
In [5]: from nltk.corpus import words

        nltk.download('words')

        correct_spellings = words.words()

        correct_spellings
```

```
[nltk_data] Downloading package words to /home/jovyan/nltk_data...
[nltk_data]   Package words is already up-to-date!
```

```
Out[5]: ['A',
         'a',
'aa',
'aal',
'aalii',
'aam',
'Aani',
'aardvark',
'aardwolf',
'Aaron',
'Aaronic',
'Aaronical',
'Aaronite',
'Aaronitic',
'Aaru',
'Ab',
'aba',
'Ababdeh',
'Ababua',
'abac',
'abaca',
'abacate',
'abacay',
'abacinate',
'abacination',
'abaciscus',
'abacist',
'aback',
'abactinal',
'abactinally',
'abaction',
'abactor',
'abaculus',
'abacus',
'Abadite',
'abaff',
'abaft',
'abaisance',
'abaiser',
'abaissed',
'abalienate',
'abalienation',
'abalone',
'Abama',
'abampere',
'abandon',
'abandonable',
'abandoned',
'abandonedly',
'abandonee',
'abandoner',
'abandonment',
'Abanic',
'Abantes',
'abaptiston',
'Abarambo',
'Abaris',
'abarthrosis',
'abarticular',
'abarticulation',
'abas',
'abase',
'abased',
'abasedly',
'abasedness',
'abasement',
'abaser',
'Abasgi',
'abash',
'abashed',
'abashedly',
'abashedness',
'abashless',
'abashlessly',
'abashment',
'abasia',
'abasic',
'abask',
'Abassin',
'abastardize',
'abatable',
'abate',
'abatement',
'abater',
'abatis',
'abatised',
'abaton',
'abator',
'abattoir',
'Abatua',
'abature',
'abave',
'abaxial',
'abaxile',
'abaze',
'abb',
'Abba',
'abbacomes',
'abbacy',
'Abbadide',
'abbas',
'abbasi',
'abbassi',
'Abbasside',
'abbatial',
'abbatical',
'abbess',
'abbey',
'abbeystede',
'Abbie',
'abbot',
'abbotcy',
'abbotnullius',
'abbotship',
'abbreviate',
'abbreviately',
'abbreviation',
'abbreviator',
'abbreviatory',
'abbreviature',
'Abby',
'abcoulomb',
'abdal',
'abdat',
'Abderian',
'Abderite',
'abdest',
'abdicable',
'abdicant',
'abdicate',
'abdication',
'abdicative',
'abdicator',
'Abdiel',
'abditive',
'abditory',
'abdomen',
'abdominal',
'Abdominales',
'abdominalian',
'abdominally',
'abdominoanterior',
'abdominocardiac',
'abdominocentesis',
'abdominocystic',
'abdominogenital',
'abdominohysterectomy',
'abdominohysterotomy',
'abdominoposterior',
'abdominoscope',
'abdominoscopy',
'abdominothoracic',
'abdominous',
'abdominovaginal',
'abdominovesical',
'abduce',
'abducens',
'abducent',
'abduct',
'abduction',
'abductor',
'Abe',
'abeam',
'abear',
'abearance',
'abecedarian',
'abecedarium',
'abecedary',
'abed',
'abeigh',
'Abel',
'abele',
'Abelia',
'Abelian',
'Abelicea',
'Abelite',
'abelite',
'Abelmoschus',
'abelmosk',
'Abelonian',
'abeltree',
'Abencerrages',
'abenteric',
'abepithymia',
'Aberdeen',
'aberdevine',
'Aberdonian',
'Aberia',
'aberrance',
'aberrancy',
'aberrant',
'aberrate',
'aberration',
'aberrational',
'aberrator',
'aberrometer',
'aberroscope',
'aberuncator',
'abet',
'abetment',
'abettal',
'abettor',
'abevacuation',
'abey',
'abeyance',
'abeyancy',
'abeyant',
'abfarad',
'abhenry',
'abhiseka',
'abhominable',
'abhor',
'abhorrence',
'abhorrency',
'abhorrent',
'abhorrently',
'abhorrer',
'abhorrible',
'abhorring',
'Abhorson',
'abidal',
'abidance',
'abide',
'abider',
'abidi',
'abiding',
'abidingly',
'abidingness',
'Abie',
'Abies',
'abietate',
'abietene',
'abietic',
'abietin',
'Abietineae',
'abietineous',
'abietinic',
'Abiezer',
'Abigail',
'abigail',
'abigailship',
'abigeat',
'abigeus',
'abilao',
'ability',
'abilla',
'abilo',
'abintestate',
'abiogenesis',
'abiogenesist',
'abiogenetic',
'abiogenetical',
'abiogenetically',
'abiogenist',
'abiogenous',
'abiogeny',
'abiological',
'abiologically',
'abiology',
'abiosis',
'abiotic',
'abiotrophic',
'abiotrophy',
'Abipon',
'abir',
'abirritant',
'abirritate',
'abirritation',
'abirritative',
'abiston',
'Abitibi',
'abiuret',
'abject',
'abjectedness',
'abjection',
'abjective',
'abjectly',
'abjectness',
'abjoint',
'abjudge',
'abjudicate',
'abjudication',
'abjunction',
'abjunctive',
'abjuration',
'abjuratory',
'abjure',
'abjurement',
'abjurer',
'abkar',
'abkari',
'Abkhas',
'Abkhasian',
'ablach',
'ablactate',
'ablactation',
'ablare',
'ablastemic',
'ablastous',
'ablate',
'ablation',
'ablatitious',
'ablatival',
'ablative',
'ablator',
'ablaut',
'ablaze',
'able',
'ableeze',
'ablegate',
'ableness',
'ablepharia',
'ablepharon',
'ablepharous',
'Ablepharus',
'ablepsia',
'ableptical',
'ableptically',
'abler',
'ablest',
'ablewhackets',
'ablins',
'abloom',
'ablow',
'ablude',
'abluent',
'ablush',
'ablution',
'ablutionary',
'abluvion',
'ably',
'abmho',
'Abnaki',
'abnegate',
'abnegation',
'abnegative',
'abnegator',
'Abner',
'abnerval',
'abnet',
'abneural',
'abnormal',
'abnormalism',
'abnormalist',
'abnormality',
'abnormalize',
'abnormally',
'abnormalness',
'abnormity',
'abnormous',
'abnumerable',
'Abo',
'aboard',
'Abobra',
'abode',
'abodement',
'abody',
'abohm',
'aboil',
'abolish',
'abolisher',
'abolishment',
'abolition',
'abolitionary',
'abolitionism',
'abolitionist',
'abolitionize',
'abolla',
'aboma',
'abomasum',
'abomasus',
'abominable',
'abominableness',
'abominably',
'abominate',
'abomination',
'abominator',
'abomine',
'Abongo',
'aboon',
'aborad',
'aboral',
'aborally',
'abord',
'aboriginal',
'aboriginality',
    'aboriginally',
    'aboriginary',
    'aborigine',
    'abort',
    'aborted',
    'aborticide',
    'abortient',
    'abortifacient',
    'abortin',
    'abortion',
    'abortional',
    'abortionist',
    'abortive',
    'abortively',
    'abortiveness',
    'abortus',
    'abouchement',
    'abound',
    'abounder',
    'abounding',
    'aboundingly',
    'about',
    'abouts',
    'above',
    'aboveboard',
    'abovedeck',
    'aboveground',
    'aboveproof',
    'abovestairs',
    'abox',
    'abracadabra',
    'abrachia',
    'abradant',
    'abrade',
    'abrader',
    'Abraham',
    'Abrahamic',
    'Abrahamidae',
    'Abrahamite',
    'Abrahamitic',
    'abraid',
    'Abram',
    'Abramis',
    'abranchial',
    'abranchialism',
    'abranchian',
    'Abranchiata',
    'abranchiate',
'abranchious',
'abrasax',
'abrase',
'abrash',
'abrasiometer',
'abrasion',
'abrasive',
'abrastol',
'abraum',
'abraxas',
'abreact',
'abreaction',
'abreast',
'abrenounce',
'abret',
'abrico',
'abridge',
'abridgeable',
'abridged',
'abridgedly',
'abridger',
'abridgment',
'abrim',
'abrin',
'abristle',
'abroach',
'abroad',
'Abrocoma',
'abrocome',
'abrogable',
'abrogate',
'abrogation',
'abrogative',
'abrogator',
'Abroma',
'Abronia',
'abrook',
'abrotanum',
'abrotine',
'abrupt',
'abruptedly',
'abruption',
'abruptly',
'abruptness',
'Abrus',
'Absalom',
'absampere',
'Absaroka',
'absarokite',
'abscess',
'abscessed',
'abscession',
'abscessroot',
'abscind',
'abscise',
'abscision',
'absciss',
'abscissa',
'abscissae',
'abscisse',
'abscission',
'absconce',
'abscond',
'absconded',
'abscondedly',
'abscondence',
'absconder',
'absconsa',
'abscoulomb',
'absence',
'absent',
'absentation',
'absentee',
'absenteeism',
'absenteeship',
'absenter',
'absently',
'absentment',
'absentmindedly',
'absentness',
'absfarad',
'abshenry',
'Absi',
'absinthe',
'absinthial',
'absinthian',
'absinthiate',
'absinthic',
'absinthin',
'absinthine',
'absinthism',
'absinthismic',
'absinthium',
'absinthol',
'absit',
'absmho',
'absohm',
'absolute',
'absolutely',
'absoluteness',
'absolution',
'absolutism',
'absolutist',
'absolutistic',
'absolutistically',
'absolutive',
'absolutization',
'absolutize',
'absolutory',
'absolvable',
'absolvatory',
'absolve',
'absolvent',
'absolver',
'absolvitor',
'absolvitory',
'absonant',
'absonous',
'absorb',
'absorbability',
'absorbable',
'absorbed',
'absorbedly',
'absorbedness',
'absorbefacient',
'absorbency',
'absorbent',
'absorber',
'absorbing',
'absorbingly',
'absorbition',
'absorpt',
'absorptance',
'absorptiometer',
'absorptiometric',
'absorption',
'absorptive',
'absorptively',
'absorptiveness',
'absorptivity',
'absquatulate',
'abstain',
'abstainer',
'abstainment',
'abstemious',
'abstemiously',
'abstemiousness',
'abstention',
'abstentionist',
'abstentious',
'absterge',
'abstergent',
'abstersion',
'abstersive',
'abstersiveness',
'abstinence',
'abstinency',
'abstinent',
'abstinential',
'abstinently',
'abstract',
'abstracted',
'abstractedly',
'abstractedness',
'abstracter',
'abstraction',
'abstractional',
'abstractionism',
'abstractionist',
'abstractitious',
'abstractive',
'abstractively',
'abstractiveness',
'abstractly',
'abstractness',
'abstractor',
'abstrahent',
'abstricted',
'abstriction',
'abstruse',
'abstrusely',
'abstruseness',
'abstrusion',
'abstrusity',
'absume',
'absumption',
'absurd',
'absurdity',
'absurdly',
'absurdness',
'absvolt',
'Absyrtus',
'abterminal',
'abthain',
'abthainrie',
'abthainry',
'abthanage',
'Abu',
'abu',
'abucco',
'abulia',
'abulic',
'abulomania',
'abuna',
'abundance',
'abundancy',
'abundant',
'Abundantia',
'abundantly',
'abura',
'aburabozu',
'aburban',
'aburst',
'aburton',
'abusable',
'abuse',
'abusedly',
'abusee',
'abuseful',
'abusefully',
'abusefulness',
'abuser',
'abusion',
'abusious',
'abusive',
'abusively',
'abusiveness',
'abut',
'Abuta',
'Abutilon',
'abutment',
'abuttal',
'abutter',
'abutting',
'abuzz',
'abvolt',
'abwab',
'aby',
'abysm',
'abysmal',
'abysmally',
'abyss',
'abyssal',
'Abyssinian',
'abyssobenthonic',
'abyssolith',
'abyssopelagic',
'acacatechin',
'acacatechol',
'acacetin',
'Acacia',
'Acacian',
'acaciin',
'acacin',
'academe',
'academial',
'academian',
'Academic',
'academic',
'academical',
'academically',
'academicals',
'academician',
'academicism',
'academism',
'academist',
'academite',
'academization',
'academize',
'Academus',
'academy',
'Acadia',
'acadialite',
'Acadian',
'Acadie',
'Acaena',
'acajou',
'acaleph',
'Acalepha',
'Acalephae',
'acalephan',
'acalephoid',
'acalycal',
'acalycine',
'acalycinous',
'acalyculate',
'Acalypha',
'Acalypterae',
'Acalyptrata',
'Acalyptratae',
'acalyptrate',
'Acamar',
'acampsia',
'acana',
'acanaceous',
'acanonical',
'acanth',
'acantha',
'Acanthaceae',
'acanthaceous',
'acanthad',
'Acantharia',
'Acanthia',
'acanthial',
'acanthin',
'acanthine',
'acanthion',
'acanthite',
'acanthocarpous',
'Acanthocephala',
'acanthocephalan',
'Acanthocephali',
'acanthocephalous',
'Acanthocereus',
'acanthocladous',
'Acanthodea',
'acanthodean',
'Acanthodei',
'Acanthodes',
'acanthodian',
'Acanthodidae',
'Acanthodii',
'Acanthodini',
'acanthoid',
'Acantholimon',
'acanthological',
'acanthology',
'acantholysis',
'acanthoma',
'Acanthomeridae',
'acanthon',
'Acanthopanax',
'Acanthophis',
'acanthophorous',
'acanthopod',
'acanthopodous',
'acanthopomatous',
'acanthopore',
'acanthopteran',
'Acanthopteri',
'acanthopterous',
'acanthopterygian',
'Acanthopterygii',
'acanthosis',
'acanthous',
'Acanthuridae',
'Acanthurus',
'acanthus',
'acapnia',
'acapnial',
'acapsular',
'acapu',
'acapulco',
'acara',
'Acarapis',
'acardia',
'acardiac',
'acari',
'acarian',
'acariasis',
'acaricidal',
'acaricide',
'acarid',
'Acarida',
'Acaridea',
'acaridean',
'acaridomatium',
'acariform',
'Acarina',
'acarine',
'acarinosis',
'acarocecidium',
'acarodermatitis',
'acaroid',
'acarol',
'acarologist',
'acarology',
'acarophilous',
'acarophobia',
'acarotoxic',
'acarpelous',
'acarpous',
'Acarus',
'Acastus',
'acatalectic',
'acatalepsia',
'acatalepsy',
'acataleptic',
'acatallactic',
'acatamathesia',
'acataphasia',
'acataposis',
'acatastasia',
'acatastatic',
'acate',
'acategorical',
'acatery',
'acatharsia',
'acatharsy',
'acatholic',
'acaudal',
'acaudate',
'acaulescent',
'acauline',
'acaulose',
'acaulous',
'acca',
'accede',
'accedence',
'acceder',
'accelerable',
'accelerando',
'accelerant',
'accelerate',
'accelerated',
'acceleratedly',
'acceleration',
'accelerative',
'accelerator',
'acceleratory',
'accelerograph',
'accelerometer',
'accend',
'accendibility',
'accendible',
'accension',
'accensor',
'accent',
'accentless',
'accentor',
'accentuable',
'accentual',
'accentuality',
'accentually',
'accentuate',
'accentuation',
'accentuator',
'accentus',
'accept',
'acceptability',
'acceptable',
'acceptableness',
'acceptably',
'acceptance',
'acceptancy',
'acceptant',
'acceptation',
'accepted',
'acceptedly',
'accepter',
'acceptilate',
'acceptilation',
'acception',
'acceptive',
'acceptor',
'acceptress',
'accerse',
'accersition',
'accersitor',
'access',
'accessarily',
'accessariness',
'accessary',
'accessaryship',
'accessibility',
'accessible',
'accessibly',
'accession',
'accessional',
'accessioner',
'accessive',
'accessively',
'accessless',
'accessorial',
'accessorily',
'accessoriness',
'accessorius',
'accessory',
'accidence',
'accidency',
'accident',
'accidental',
'accidentalism',
'accidentalist',
'accidentality',
'accidentally',
'accidentalness',
'accidented',
'accidential',
'accidentiality',
'accidently',
'accidia',
'accidie',
'accinge',
'accipient',
'Accipiter',
'accipitral',
'accipitrary',
'Accipitres',
'accipitrine',
'accismus',
'accite',
'acclaim',
'acclaimable',
'acclaimer',
'acclamation',
'acclamator',
'acclamatory',
'acclimatable',
'acclimatation',
'acclimate',
'acclimatement',
'acclimation',
'acclimatizable',
'acclimatization',
'acclimatize',
'acclimatizer',
'acclimature',
'acclinal',
'acclinate',
'acclivitous',
'acclivity',
'acclivous',
'accloy',
'accoast',
'accoil',
'accolade',
'accoladed',
 'accolated',
 'accolent',
 'accolle',
 'accombination',
 'accommodable',
 'accommodableness',
 'accommodate',
 'accommodately',
 'accommodateness',
 'accommodating',
 'accommodatingly',
 'accommodation',
 'accommodational',
 'accommodative',
 'accommodativeness',
 'accommodator',
 'accompanier',
 'accompaniment',
 'accompanimental',
 'accompanist',
 'accompany',
 'accompanyist',
 'accompletive',
 'accomplice',
 'accompliceship',
 'accomplicity',
 'accomplish',
 'accomplishable',
 'accomplished',
 'accomplisher',
 'accomplishment',
 'accomplisht',
 'accompt',
 'accord',
 'accordable',
 'accordance',
 'accordancy',
 'accordant',
 ...]
```

1.2.1 Question 9

For this recommender, your function should provide recommendations for the three default words provided above using the following distance metric:

Jaccard distance on the trigrams of the two words.

This function should return a list of length three: `['cormulent_reccomendation', 'incendenece_reccomendation', 'validrate_reccomendation']`.

```
In [36]: import warnings

         warnings.filterwarnings('ignore')
```

```
In [84]: def answer_nine(entries=['cormulent', 'incendenece', 'validrate']):

             from nltk.metrics import jaccard_distance
             from nltk import ngrams
             from operator import itemgetter

             r1 = [x for x in correct_spellings if x[0]==entries[0][0]]
             r2 = [x for x in correct_spellings if x[0]==entries[1][0]]
             r3 = [x for x in correct_spellings if x[0]==entries[2][0]]

             ng3_1 = set(nltk.ngrams(entries[0],3))
             ng3_2 = set(nltk.ngrams(entries[1],3))
             ng3_3 = set(nltk.ngrams(entries[2],3))

             jd1 = list()
             jd2 = list()
             jd3 = list()

             for word in r1:
                 jd1.append([nltk.jaccard_distance(ng3_1,set(nltk.ngrams(word,3))), word])

             for word in r2:
                 jd2.append([nltk.jaccard_distance(ng3_2,set(nltk.ngrams(word,3))), word])

             for word in r3:
                 jd3.append([nltk.jaccard_distance(ng3_3,set(nltk.ngrams(word,3))), word])

             jd1_sorted = sorted(jd1, key = itemgetter(0))
             jd2_sorted = sorted(jd2, key = itemgetter(0))
             jd3_sorted = sorted(jd3, key = itemgetter(0))

             return [jd1_sorted[0][1],jd2_sorted[0][1],jd3_sorted[0][1]]

         answer_nine()
```

```
Out[84]: ['corpulent', 'indecence', 'validate']
```

1.2.2 Question 10

For this recommender, your function should provide recommendations for the three default words provided above using the following distance metric:

Jaccard distance on the 4-grams of the two words.

This function should return a list of length three: `['cormulent_reccomendation', 'incendenece_reccomendation', 'validrate_reccomendation']`*.*

```
In [88]: def answer_ten(entries=['cormulent', 'incendenece', 'validrate']):

             from nltk.metrics import jaccard_distance
             from nltk import ngrams
             from operator import itemgetter

             rA = [x for x in correct_spellings if x[0]==entries[0][0]]
             rB = [x for x in correct_spellings if x[0]==entries[1][0]]
             rC = [x for x in correct_spellings if x[0]==entries[2][0]]

             ng3_A = set(nltk.ngrams(entries[0],4))
             ng3_B = set(nltk.ngrams(entries[1],4))
             ng3_C = set(nltk.ngrams(entries[2],4))

             jdA = list()
             jdB = list()
             jdC = list()

             for word in rA:
                 jdA.append([nltk.jaccard_distance(ng3_A,set(nltk.ngrams(word,4))), word])

             for word in rB:
                 jdB.append([nltk.jaccard_distance(ng3_B,set(nltk.ngrams(word,4))), word])

             for word in rC:
                 jdC.append([nltk.jaccard_distance(ng3_C,set(nltk.ngrams(word,4))), word])

             jdA_sorted = sorted(jdA, key = itemgetter(0))
             jdB_sorted = sorted(jdB, key = itemgetter(0))
             jdC_sorted = sorted(jdC, key = itemgetter(0))


             return [jdA_sorted[0][1],jdB_sorted[0][1],jdC_sorted[0][1]]

         answer_ten()
```

```
Out[88]: ['cormus', 'incendiary', 'valid']
```

1.2.3 Question 11

For this recommender, your function should provide recommendations for the three default words provided above using the following distance metric:

Edit distance on the two words with transpositions.

This function should return a list of length three: `['cormulent_reccomendation', 'incendenece_reccomendation', 'validrate_reccomendation']`.

```
In [93]: def answer_eleven(entries=['cormulent', 'incendenece', 'validrate']):
    from operator import itemgetter

    rX = [x for x in correct_spellings if x[0]==entries[0][0]]
    rY = [x for x in correct_spellings if x[0]==entries[1][0]]
    rZ = [x for x in correct_spellings if x[0]==entries[2][0]]

    dldA = list()
    dldB = list()
    dldC = list()

    for word in rX:
        dldA.append([nltk.edit_distance(entries[0],word, transpositions = True), word])

    for word in rY:
        dldB.append([nltk.edit_distance(entries[1],word, transpositions = True), word])

    for word in rZ:
        dldC.append([nltk.edit_distance(entries[2],word, transpositions = True), word])

    dldA_sorted = sorted(dldA, key = itemgetter(0))
    dldB_sorted = sorted(dldB, key = itemgetter(0))
    dldC_sorted = sorted(dldC, key = itemgetter(0))


    return [dldA_sorted[0][1], dldB_sorted[0][1], dldC_sorted[0][1]]

answer_eleven()
```

```
Out[93]: ['corpulent', 'intendence', 'validate']
```

```
In [ ]:
```