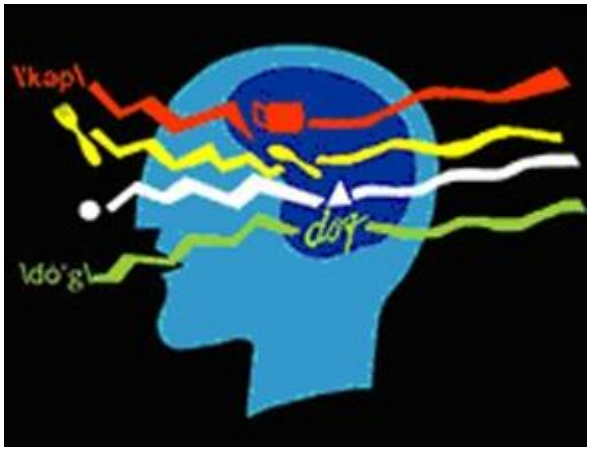
Topic 2 (Pt.2):

Python Natural Language Processing Toolkit



Content

- **What we will learn today:**
  - Intro to Python NLTK
  - NLTK Text Processing
  - Corpus vs Corpora
  - Accessing Text Corpora with NLTK
  - Concordance, Text Similarity & Common Contexts
  - Plotting Word Distribution Graph
  - Generating Random Text
  - Counting Vocabulary
  - Token vs Types, Lexicon vs Grammar, Tokenization
  - Zipf Law & Frequency Distribution
  - Storing Large Files

Python Natural Language Toolkit

- NLTK is a leading platform for building Python programs to work with human language data.
- Provides the following easy-to-use interfaces :
  - Over 50 corpora and lexical resources such as WordNet
  - a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing and semantic reasoning
- Link to NLTK book : http://www.nltk.org/book/

Installing NLTK

- For Windows, if Python is not installed (32-bit binary installation)
  - Install Python: https://www.python.org/downloads/release/python-374/
    - (avoid the 64-bit versions)
  - Install Numpy (optional):

    https://pypi.python.org/pypi/numpy
  - Install NLTK: https://pypi.python.org/pypi/nltk

NLTK Data

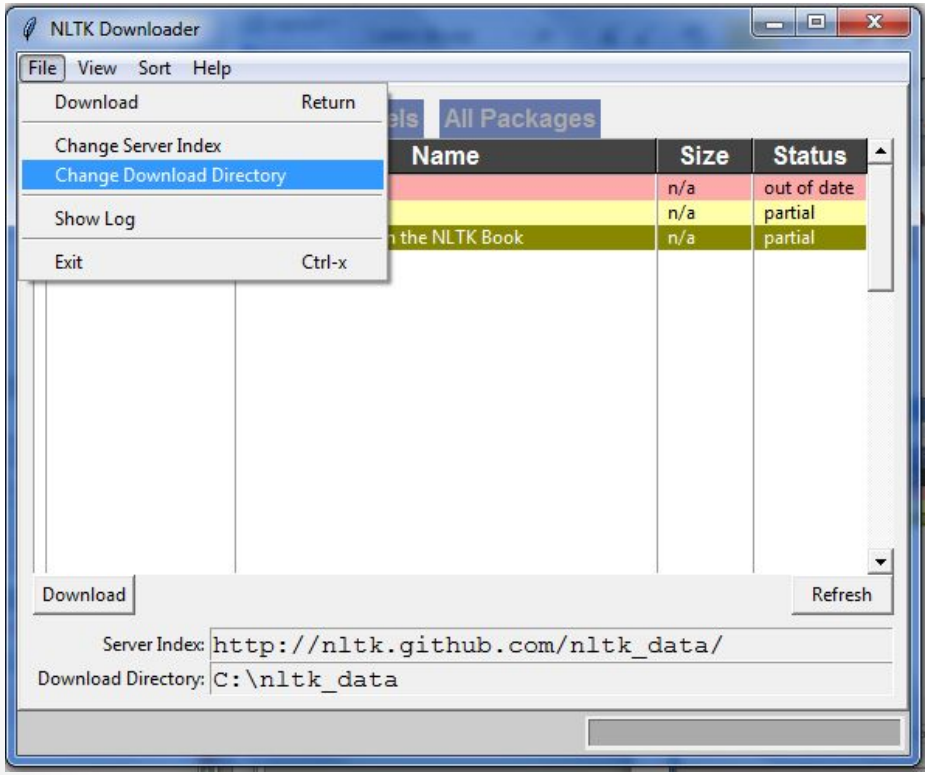
- Installing nltk data
  - Comes with many corpora, toy grammars, trained models, etc. : http://nltk.org/nltk_data/
- Steps:
  - Install NLTK (http://nltk.org/install.html)
  - Use NLTK's data downloader for:
    - Individual data packages
    - entire collection (using "all")
    - data required for the examples and exercises in the book (using "book")
    - corpora and no grammars or trained models (using "all-corpora").

NLTK Data Download

- Run the Python interpreter and type the commands:

```
>>> import nltk
>>> nltk download()
SyntaxError: invalid syntax
>>> nltk.download()
showing info http://nltk.github.com/nltk_data/
```

NLTK Data Download



- A new window pops up, showing the NLTK Downloader.
- Click on the File menu, select Change Download Directory.
  - Set this to default C:\nltk_data (Windows), or /usr/share/nltk_data (Mac, Unix).
  - Select the packages or collections you want to download (choose 'book').

Testing NLTK Data

```
>>> from nltk.corpus import brown
>>> brown.words()
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

Getting Started with NLTK

- Practical work in Natural Language Processing typically uses large bodies of linguistic data or corpora
  - What are some useful text **corpora** and **lexical resources** that can be accessed with Python?
  - Which **Python constructs** are most helpful for this work?
  - How do we **avoid repeating** ourselves when writing Python code?

NLTK: Text Processing

- Sentiment analysis
- Spam filtering
- Plagiarism Detection/Document Similarity
- Document Categorization/Topic Detection
- Phrase Extraction/Summarization
- Smart Search
- Frequency Analysis
- Sentence & Word Tokenization
- Part-of-speech Tagging
- Chunking and named Entity Recognition
- Text Classification

What is a Corpus?

- A single large collection of text used in linguistics research, may be in the form of written or spoken material
- Provides grammarians, lexicographers, and researchers in NLP with better descriptions of a language
- Available in different formats such as raw text, transcriptions of conversations, labelled text (e.g., tagged with part-of-speech), parsed phrases, phone conversations, ...

What is Corpora (plural)?

- Multiple collections of text (i.e., spoken or written)
  - Computer-processable corpora allow linguists to adopt the principle of total accountability, retrieve word occurrences or structures of randomly selected samples
  - Provide lexical information, morpho-syntactic information, semantic information and pragmatic information.
  - Used to develop NLP tools such as spell-checking, grammar-checking, speech-recognition, text-to-speech synthesis, machine translation, etc…
  - Available in monolingual (1 language), bilingual (2 languages) or multilingual (multiple languages)

Example list of Popular Corpora

- British National Corpus (BNC)
  - 100 million word collection of samples of written and spoken language from a wide range of sources
  - http://www.natcorp.ox.ac.uk/corpus/index.xml
- Brown Corpus
- Child Language Data Exchange Systems (CHILDES)
  - The child language component of the TalkBank system for sharing and studying child conversational interactions.
    - https://childes.talkbank.org/

Example list of Popular Corpora

- Penn Treebank
  - Annotated text for linguistic structure with syntactic and semantic information (a *bank* of linguistic *trees)* as well as part-of-speech
  - https://catalog.ldc.upenn.edu/LDC99T42

The Brown Corpus

- A general corpus (text collection) in the field of corpus linguistics consisting of running text of edited English
- Contains 500 samples of English-language text, totalling roughly 1,014,312 words, compiled from works published in the United States in 1961.
- The Corpus is divided into 500 samples of 2000+ words each distributed across 15 genres
- Examples of genres:
  - Reportage (44 texts)
  - Editorial (27 texts)
  - Reviews (17 texts)
  - Religion (17 texts) , etc…

Accessing Text Corpora

- Load some texts from nltk.book import *.
- This says "from NLTK's book module, load all items."

```
>>> from nltk.book import *
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
>>>
```

Accessing Text Corpora

- Find out about text :

```
>>> text1
<Text: Moby Dick by Herman Melville 1851>
>>> text2
<Text: Sense and Sensibility by Jane Austen 1811>
>>>
```

- Searching Text
  - Examine the context of a text based on concordance, text similarity and common contexts.

Concordance

- **Concordance** shows us every occurrence of a given word, together with some context & permits us to see words in context.
- Results display the occurrences of "monstrous" in different contexts of sentences in text1
- We can observe that monstrous occurred in contexts such as the ___ pictures and the ___ size .
- **Concordance** for the word "mystery"?

```
>>> text1.concordance("monstrous")
Building index...
Displaying 11 of 11 matches:
ong the former , one was of a most monstrous size . ... This came towards us ,
ON OF THE PSALMS . " Touching that monstrous bulk of the whale or ork we have r
ll over with a heathenish array of monstrous clubs and spears . Some were thick
d as you gazed , and wondered what monstrous cannibal and savage could ever hav
that has survived the flood ; most monstrous and most mountainous ! That Himmal
they might scout at Moby Dick as a monstrous fable , or still worse and more de
th of Radney .'" CHAPTER 55 Of the monstrous Pictures of Whales . I shall ere l
ing Scenes . In connexion with the monstrous pictures of whales , I am strongly
ere to enter upon those still more monstrous stories of them which are to be fo
ght have been rummaged out of this monstrous cabinet there is no telling . But
of Whale - Bones ; for Whales of a monstrous size are oftentimes cast up dead u
>>>
```

Text Similarity

- **Text similarity** find other words appearing in a similar range of contexts using similar
- **Text similarity** for the word "mystery"?

```
>>> text1.similar("monstrous")
Building word-context index...
subtly impalpable pitiable curious imperial perilous trustworthy
abundant untoward singular lamentable few maddens horrible loving lazy
mystifying christian exasperate puzzled
>>> text2.similar("monstrous")
Building word-context index...
very exceedingly so heartily a great good amazingly as sweet
remarkably extremely vast
>>>
```

Examine Common Context

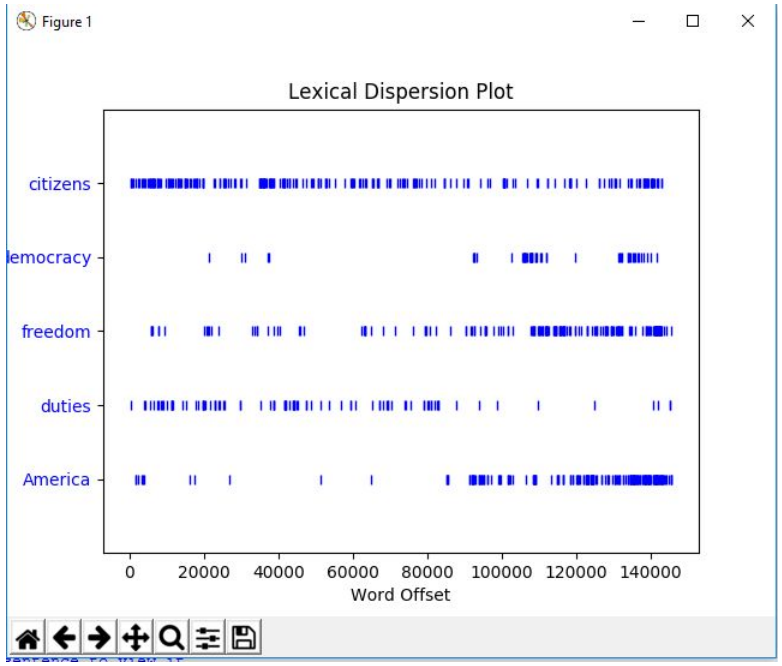
- The term **common_contexts** allows us to examine just the contexts that are shared by two or more words, such as monstrous and very.
- Enclose these words by square brackets as well as parentheses, and separate them with a comma.
- What is the **common contexts** for "comfortable" and "more"?

```
>>> text2.common_contexts(["monstrous", "very"])
be_glad am_glad a_pretty is_pretty a_lucky
>>>
```

Plotting Graphs

- Using *dispersion plot* for text dispersion
  - Determine the location of a word in the text and its positional information. Each stripe represents an instance of a word, and each row represents the entire text.

```
>>> text4.dispersion_plot(["citizens", "democracy", "freedom", "duties", "America"])
>>>
```



Counting Vocabulary

- Use len() in Python & NLTK to find the length or size of a string (i.e, no. of character/words tokens in a text/string or sentences in texts)

```
>>> from nltk.book import *
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908

>>> len(text5)
45010
```

Important Terms in Text Processing

- Token vs type
- Lexicon and Grammar
- Tokenization
- Zipf Law
- Frequency distribution

Tokens vs Types

- A *token* is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing.
- A *type* is the class of all tokens containing the same character sequence.

  "The boy in the blue shirt is eating the chocolate ice-cream in the kitchen"
- 14 word *tokens*
- 12 word *types* (unique words)

Lexicon and Grammar

- Lexicon is a dictionary of word definitions

Lexicon

word	category	semantics
cat	Noun	λx • feline(x)
chased	Verb	λxy • x Λ y Λ chased(x, y)
large	Adjective	λx • largesize(x)
rat	Noun	λx • rodent(x)
the	Article	$\exists_1$ ⟨gensym⟩

- Grammar is a set of syntax rules

Grammar

Syntactic rule	Semantic rule
Sentence → NounPhrase, VerbPhrase	apply VerbPhrase (NP)
VerbPhrase → Verb, NounPhrase	apply Verb (NounPhrase)
NounPhrase → Article, Noun	apply Noun (Article)
NounPhrase → Article, Adjective, NounPhrase	apply Adjective (Article) Λ apply Noun (Article)

Tokenization

- Given a character sequence (i.e., word) or word sequence (i.e., sentence), tokenization is the task of chopping it up into pieces including:
  - Removing unnecessary characters such as punctuations and symbols in Python using the **re** module
    - **re.sub("[$!? .,#@]","",sent)**
  - Normalization – changing all tokens into standard case (i.e., all lower case tokens)
    - sent.lower()

Text Tokenization (NLTK)

- The task of chopping strings (i.e., text) into pieces based on a certain boundary
- A text is as sequence of words and punctuations

```
>>> from nltk.tokenize import sent_tokenize
>>> sent_tokenize("Salaam Python. This is your mother, Anaconda")
['Salaam Python.', 'This is your mother, Anaconda']
>>> sent_tokenize("Salaam Python. This is your mother, Anaconda")[0]
'Salaam Python.'

>>> sent = sent_tokenize("Salaam NLP'ers. How are you today?")
>>> sent[0]
"Salaam NLP'ers."
```

```
>>> sent = "Natural Language Processing"
>>> sent.split()
['Natural', 'Language', 'Processing']
>>> sent.split()[0]
'Natural'
```

Text Tokenization (cont)

- Tweet tokenization

```
>>> from nltk.tokenize import TweetTokenizer
>>> token = TweetTokenizer()
>>> tweet = "This is a cooool #smiley: :-) :-P <3 and some arrows < > -> <--"
>>> token.tokenize(tweet)
['This', 'is', 'a', 'cooool', '#smiley', ':', ':-)', ':-P', '<3', 'and', 'some',
 'arrows', '<', '>', '->', '<--']
```

Tokenization

- What are the right tokens to use?
  - Language specific (different language supports different char set)
    - Malay (Latin char sets, same like English)
    - Arabic (Arabic char sets)
    - German (German char sets)
    - Swahili (Swahili char sets)

Word Selection Example

- Some words are informative and may contain many characteristics
- Example: find the longest word in a text that are more than 15 characters long
  - Express word of interest using Mathematical notation:

    $\{ w \mid w \in V \text{ and } P(w) \}$
  - The set of all *w* such that *w* is an element of V and w has property P (in this case *P(w)* is len > 15)

```
>>> V = set(text1)
>>> for w in V:
        if len(w) > 15:
            print(w)
```

Method 1

```
>>> [w for w in V if len(w) > 15]
```

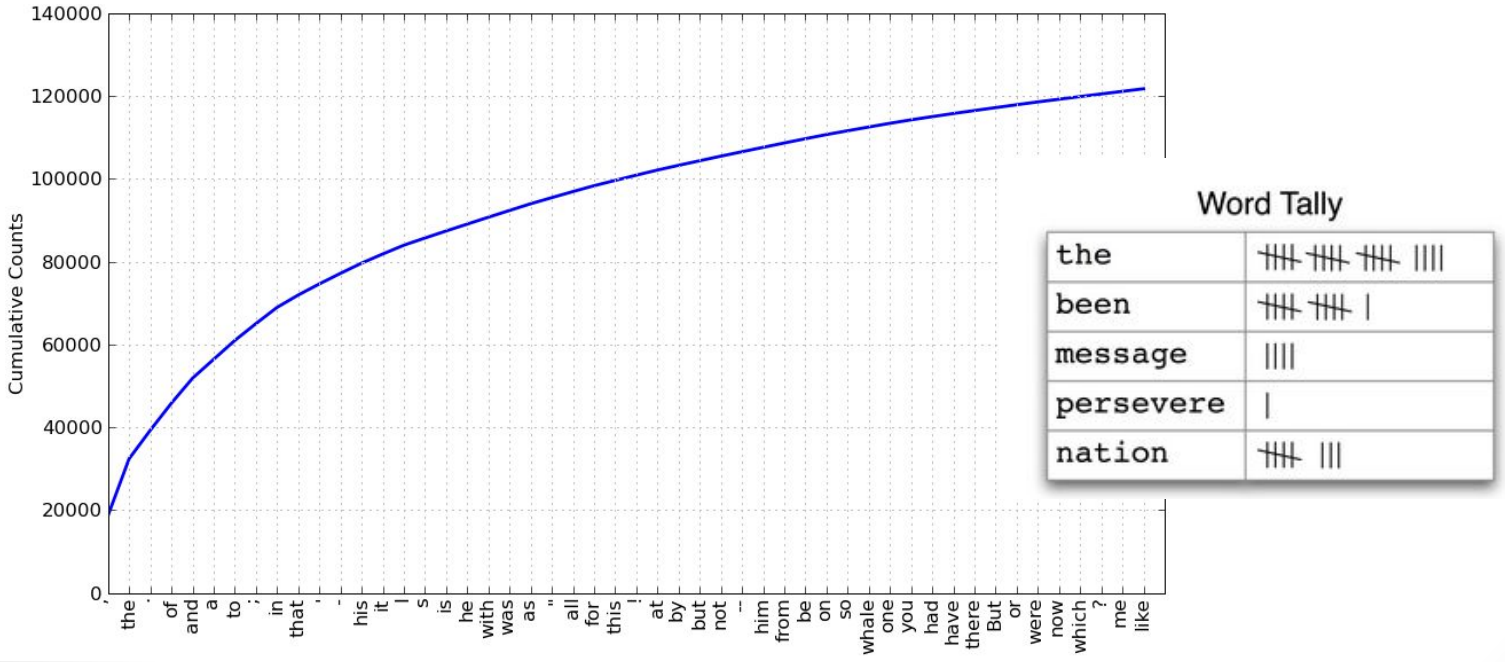
Method 2

Zipf Law

- Zipf's law states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table.
  - The **most frequent word** will **occur approximately twice as often as the second most frequent** word, **three times as often** as **the third most frequent** word, etc.
  - Most standard texts shall conform to the Zipf law

Frequency Distribution

- How to automatically identify the words in a text that are most informative about the topic and genre of the text?
  - Keep a tally



Word Tally

the	𝍸 𝍸 𝍸 ||||
been	𝍸 𝍸 |
message	||||
persevere	|
nation	𝍸 |||

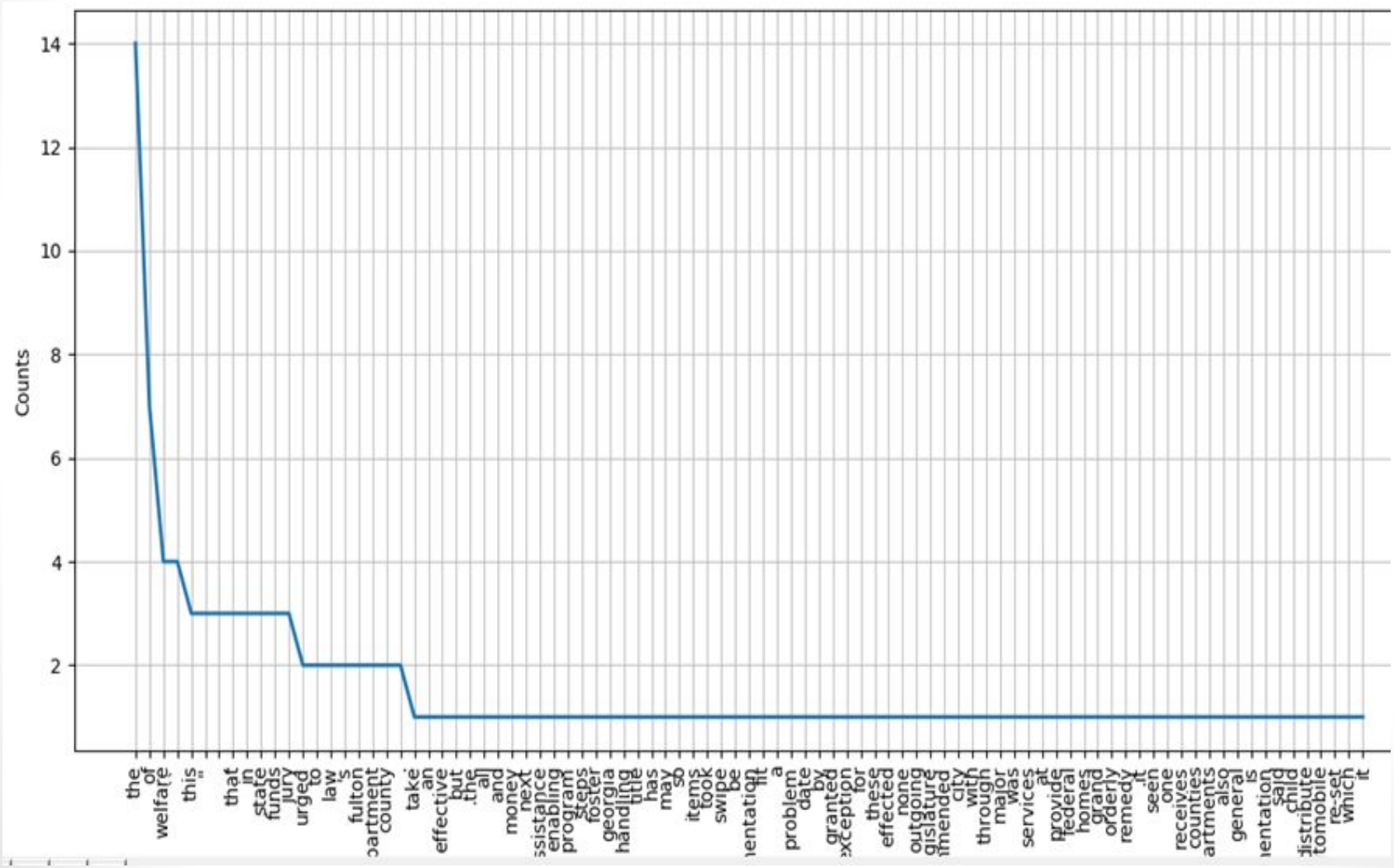
Plotting Frequency Distribution with NLTK(1)

- To use FreqDist() in the following example, you first need to install matplotlib package.
- Use pip to install through your pip working directory :

  C:\Python36\Scripts> pip install –U matplotlib
- For Python version 3.5++, this will also automatically install other scientific packages like numpy

```
import matplotlib
import nltk
from nltk.tokenize import word_tokenize
from nltk.probability import *
from nltk.corpus import brown

sent = brown.sents()[10:15]#select sentence 10-15 of Brown corpus
para = ""

for x in range(len(sent)): #for each sentence list:
    para += ' '.join(sent[x]) #join words, append each sentence to form paragraph

fdist = FreqDist(word.lower() for word in word_tokenize(para))
fdist.plot()
```

Plotting Frequency Distribution with NLTK(1)

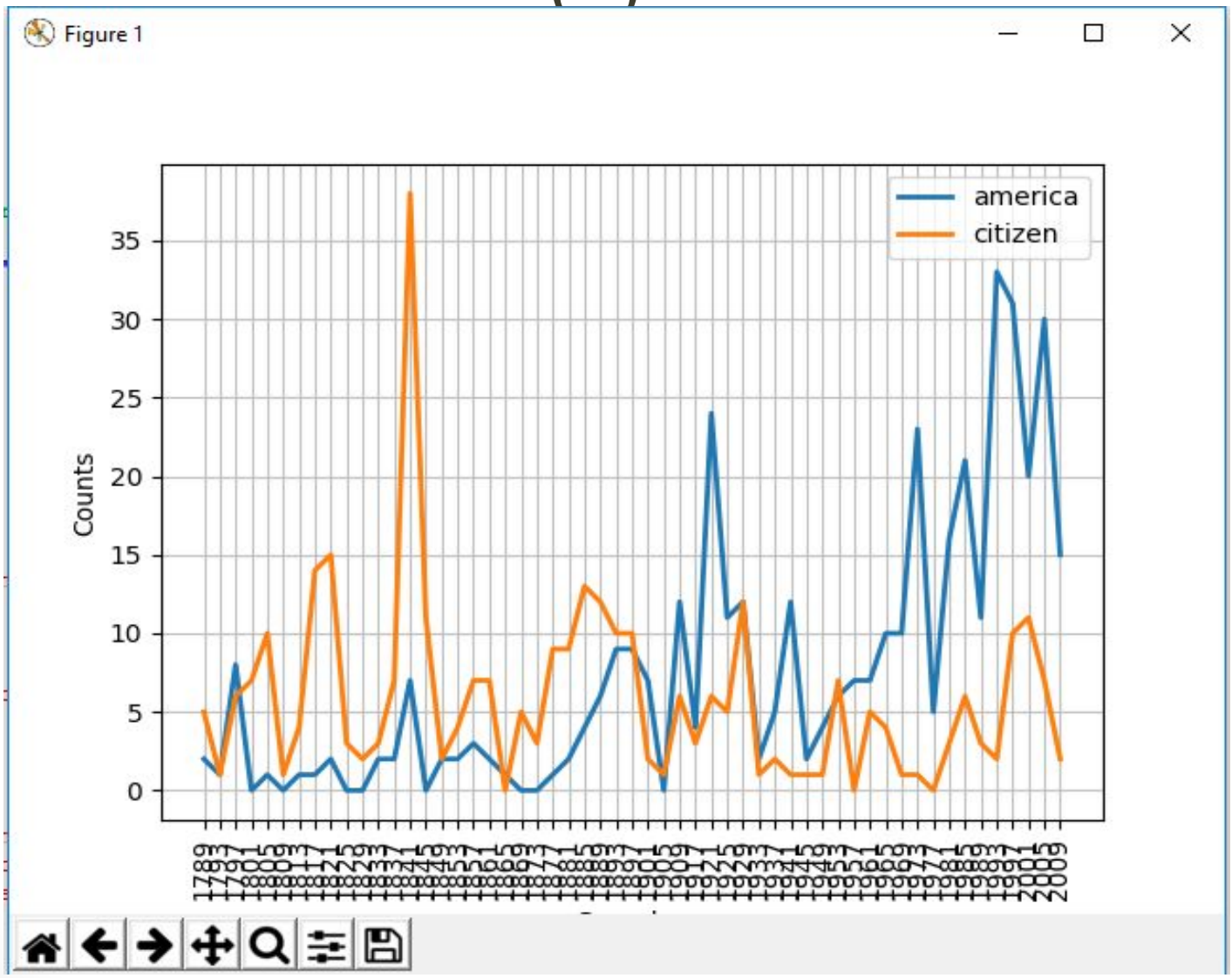


Plotting Frequency Distribution with NLTK(2)

- To use nltk.ConditionalFreqDist() in the following example, you also need the matplotlib package.

```
>>> from nltk.corpus import inaugural
>>> cfd = nltk.ConditionalFreqDist(
        (target, fileid[:4])
        for fileid in inaugural.fileids()
        for w in inaugural.words(fileid)
        for target in ['america', 'citizen']
        if w.lower().startswith(target))

>>> cfd.plot()
```

Plotting Frequency Distribution with NLTK (2)



Storing Large Files in Python

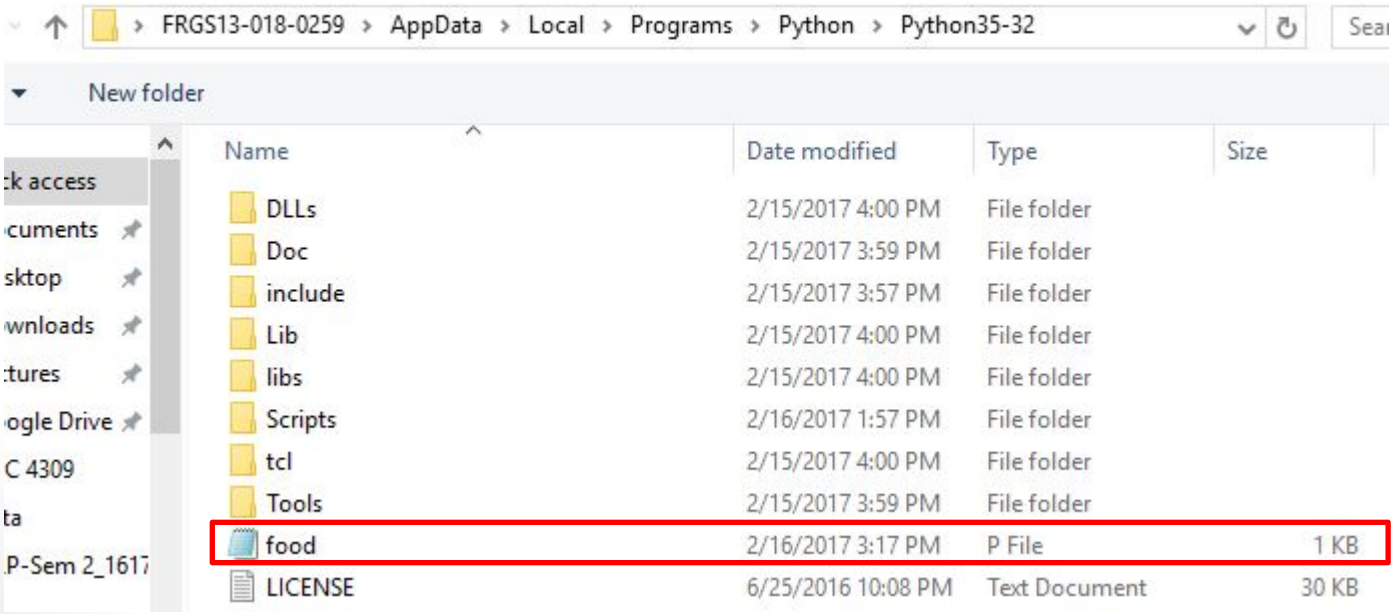
- Texts tend to be large in size and requires huge memory space, thus most files are stored in compressed forms (e.g., the texts & corpora in NLTK data)
- To efficiently store large files in Python in compressed form, we can use the functions pickle() and gzip().

Using pickle(1)

- *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation.
- pickle.dump() - Write a pickled representation of *obj* to the open file object *file*. Structure is reserved.

```
>>> import pickle  # Save a dictionary into a pickle file.
>>> favorite_food = { "roti canai": 1.00, "satay": 0.80, "pizza" : 9.99, "durian" : 10.00 }
>>> pickle.dump( favorite_food, open( "food.p", "wb" ) )
```

Store data in binary



Using pickle(2)

- Pickling in binary (“wb” & “rb” files) can save space and improve efficiency

```
€]q (X
    roti canaiq]G?ð      X]    durianq G@$      X]    satayq]G?
é™™™™™™šX]    pizzaq]G@#úáG®]{u.
```

Pickled data saved in binary format

- pickle.load()

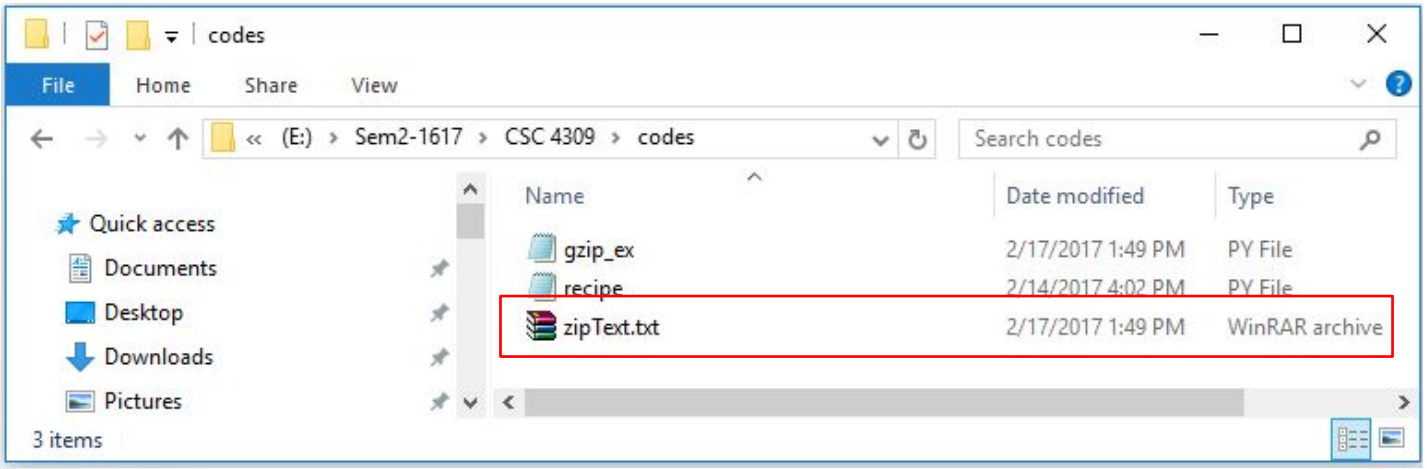
```
# Load the dictionary back from the pickle file.

>>> favorite_food = pickle.load( open( "food.p", "rb" ) )
>>> favorite_food
{'roti canai': 1.0, 'durian': 10.0, 'satay': 0.8, 'pizza': 9.99}
```

Data structure is preserved in dictionary format when loaded (unpickled)

Using gzip(1)

- The zip() function is used to save memory by only generating the elements of the iterator (such as list) as you need them, rather than putting it all into memory at once.
- Example of how to create a compressed GZIP file:

```
import os
import gzip

os.chdir('e:/sem2-1617/CSC 4309/codes/')

content = b"Lots of content here" #the 'b' prefix here means bytes, not ASCII or Unicode string
with gzip.open('zipText.txt.gz', 'wb') as f:
    f.write(content)
    f.close()
```



Using gzip(2)

- Example of how to read a compressed file:

```
import os
import gzip

os.chdir('e:/sem2-1617/CSC 4309/codes/')

with gzip.open('zipText.txt.gz', 'rb') as f:
    file_content = f.read()
    print(file_content)
    f.close()
```

Output:

```
============== RESTART: E:/Sem2-1617/CSC 4309/codes/gzip_ex.py ==============
b'Lots of content here'
>>>
```

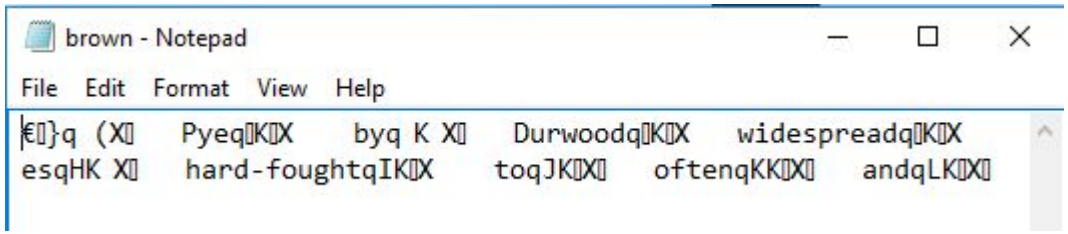
Using pickle with NLTK(1)

- Example for pickle():

```
import os
import nltk
import pickle
from nltk.corpus import brown

os.chdir('e:/sem2-1617/CSC 4309/data/')

sent = brown.sents()[0:5] #get sentence 1-5 in Brown corpus
#output of sent are lists of words, 1 list for each sentence

#use pickle to store data in dictionary format in binary file
diction = {}
for x in range(len(sent)):
    for word in sent[x]:
        #print(word)
        if word not in diction:
            diction[word] = 1
        else:
            diction[word] += 1

pickle.dump(diction,open("brown.p","wb")) #store the dictionary in binary

#Load data from pickled binary file
diction = pickle.load(open("brown.p","rb")) #read the dictionary from binary
print(diction)
```

Using pickle with NLTK(2)

- Output:




```
{'ambiguous': 1, 'size': 1, 'and': 7, 'accepted': 1, 'modernizing': 1, 'act': 1, 'did': 1, '
Durwood': 1, 'was': 2, 'often': 1, '``': 6, 'many': 1, 'commented': 1, 'irregularities': 1,
'reports': 2, 'are': 2, 'have': 1, 'relative': 1, 'improving': 1, ',': 4, 'them': 2, 'studie
d': 1, 'inadequate': 1, 'governments': 1, 'purchasing': 1, 'jury': 4, 'Pye': 1, 'in': 2, 'Al
len': 1, "''": 6, 'of': 8, 'election': 2, 'among': 1, 'practices': 1, 'Jr.': 1, 'The': 3, 'e
nd': 1, '.': 5, 'to': 4, 'both': 1, 'find': 1, 'outmoded': 1, 'the': 9, 'handful': 1, 'Fulto
n': 3, 'inure': 1, 'which': 3, 'topics': 1, 'received': 1, 'charged': 1, 'Superior': 1, 'vot
ers': 1, 'operated': 1, 'Mayor-nominate': 1, 'on': 1, 'laws': 2, 'investigate': 1, 'possible
': 1, 'or': 1, 'September-October': 1, 'well': 1, 'that': 2, 'city': 1, 'other': 1, 'grand':
1, 'such': 1, 'revised': 1, "Georgia's": 1, 'best': 1, 'Judge': 1, 'considering': 1, 'follow
': 1, 'had': 1, 'this': 1, 'Ivan': 1, 'registration': 1, 'it': 2, 'number': 2, 'widespread':
1, 'been': 1, 'term': 1, 'Only': 1, 'legislators': 1, 'recommended': 1, 'Atlanta': 1, 'a': 2
, 'It': 1, 'Court': 1, 'interest': 2, 'said': 3, 'these': 1, 'departments': 1, 'generally':
1, 'won': 1, 'primary': 1, 'County': 1, 'by': 2, 'hard-fought': 1}
```

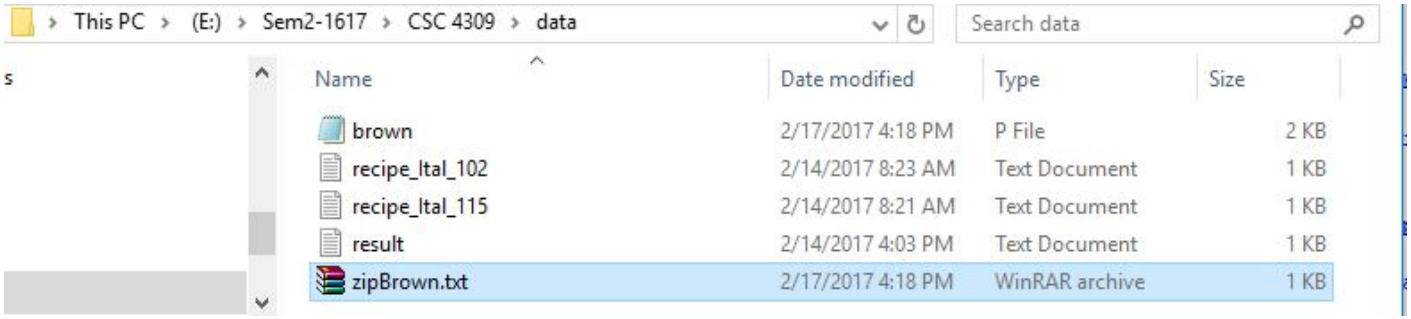
Using gzip with NLTK(1)

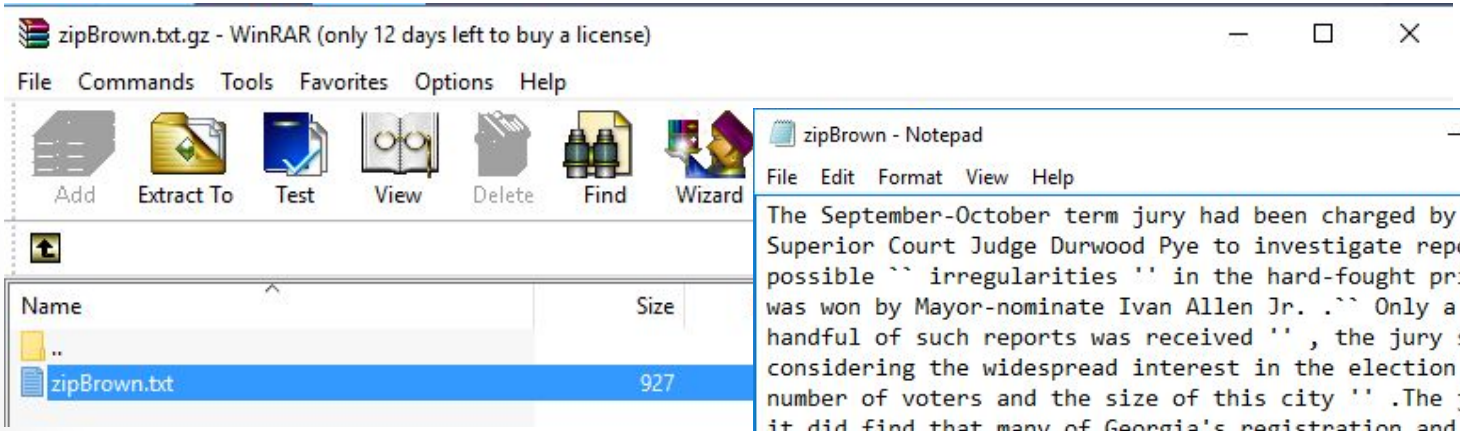
- Example for gzip():

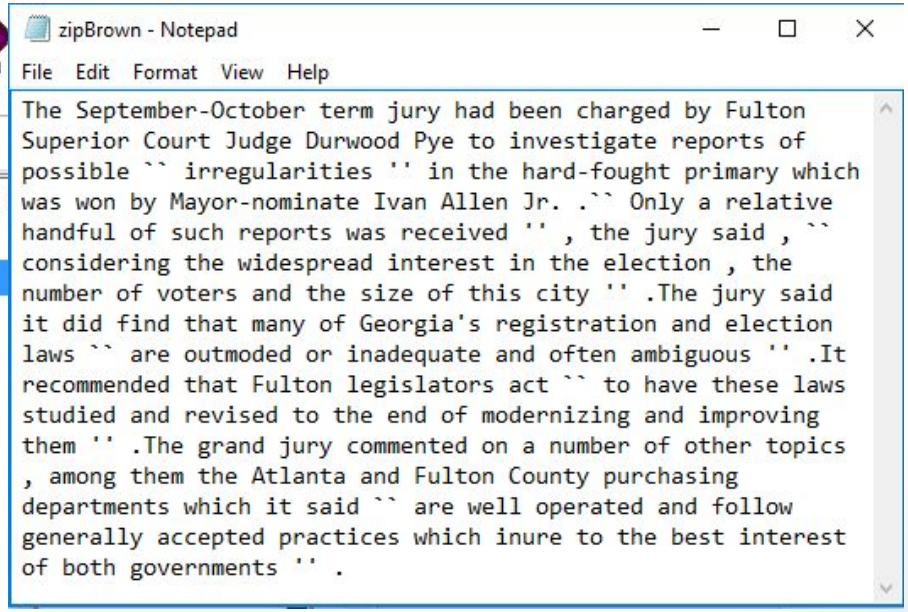
```
import os
import nltk
import gzip
from nltk.corpus import brown

with gzip.open('zipBrown.txt.gz', 'wb') as f: #store the text in binary
    for x in range(len(sent)): #for each sentence list, join words to form a string
        para = ' '.join(sent[x]) #assign output to para
        print(para)
        byte_para = str.encode(para)
        f.write(byte_para)#save paragraph of Brown text in compressed format
        type(byte_para)
    f.close()

with gzip.open('zipBrown.txt.gz', 'rb') as f:
    file_content = f.read()
    print(file_content)
    f.close()
```

Using gzip with NLTK(2)

- Output:







Exercise 2: NLTK

- Create a function in Python that accepts 2 arguments called percent (word, text) that calculates how often a given word occurs in a text.
- Use any text in the nltk book (text1, text2, etc….) or any text file (if nltk is not available)

  1) Return the value of count for the selected text

  2) Return the percentage of counts out of the overall selected text

  3) Find words in the selected text that contains 5 characters or less

  4) Print the results in an output file named nltk_out.dat