Chatbot - Scott Berry

Part 1 - Exploratory Data Analysis

How many samples/rows in the dataset?

-- There are 500 samples of science questions with multiple choices. The dataset provides the correct answer as well.

-- There are 1326 samples of factual free response science answers.

How many empty rows (e.g. missing text entries)?

-- There are no missing entries.

What is your source and target?

-- There are two different types of datasets:

---- The first is question > answer in the form of multiple choice

---- The second ii statement > label/description in the form of factual statements

Part 2 - Preprocessing

In [11]:

```
import warnings

import pandas as pd
import vec as vec

warnings.filterwarnings('ignore')
from nltk import collections
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, LancasterStemmer, SnowballStemmer
from nltk.tokenize import sent_tokenize, word_tokenize

# load openbook_txt
with open("OpenBookQA/data/OpenBookQA-V1-Sep2018/Data/Main/openbook.txt") as openbook_txt:
    openbook = openbook_txt.read()
with open("OpenBookQA/data/OpenBookQA-V1-Sep2018/Data/Main/openbook.txt") as openbook_txt:
    openbook_lines = openbook_txt.readlines()

def normalize(text):
    tokenized_sentences = [word_tokenize(t) for t in sent_tokenize(openbook)]
    words = [word for sentence in tokenized_sentences for word in sentence]
    words = [word.lower() for word in words if word.isalpha()]
    stop_words = stopwords.words('english')
    filtered_words = [w for w in words if w not in stop_words]
    return filtered_words, words

# tokenize words
filtered_words, words = normalize(openbook)
# top 20 words
word_counts = collections.Counter(filtered_words)
print(word_counts.most_common(20))
```

```
[('water', 124), ('used', 109), ('object', 105), ('causes', 88), ('environment', 81), ('energy', 76), ('food',
74), ('source', 68), ('animal', 66), ('increases', 66), ('animals', 64), ('something', 60), ('light', 57), ('h
eat', 53), ('earth', 51), ('cause', 51), ('increase', 50), ('plant', 49), ('organism', 48), ('example', 46)]
```

These frequent words indicate that the natural world and how matter interacts will be described by the dataset.

In [12]:

```
import nltk

# length of word, length of sentence,  lexical diversity
filtered_chars = [list(word) for word in filtered_words]
flattened_chars = [item for sublist in filtered_chars for item in sublist]
word_length = len(flattened_chars) / len(filtered_words)
print("Length of each word:", word_length)
sentence_length = len(words) / len(openbook_lines)
print("Length of each sentence:", sentence_length)
print("Lexical Diversity:")
```

```
def lexical_diversity(text):
    return len(text) / len(set(text))
for cat in nltk.corpus.brown.categories():
    print(cat, (lexical_diversity(cat) / 10))
```

```
Length of each word: 6.5051346100471825
Length of each sentence: 9.334841628959277
Lexical Diversity:
adventure 0.1125
belles_lettres 0.2
editorial 0.1125
fiction 0.11666666666666667
government 0.125
hobbies 0.11666666666666667
humor 0.1
learned 0.11666666666666667
lore 0.1
mystery 0.11666666666666667
news 0.1
religion 0.11428571428571428
reviews 0.11666666666666667
romance 0.1
science_fiction 0.16666666666666669
```

In [13]:

```
from IPython.display import display
from nltk import WordNetLemmatizer
import pandas as pd


sample = """
"A bee is a pollinating animal"
"A bird is a pollinating animal"
"An electrical conductor is a vehicle for the flow of electricity"
"An example of a change in the Earth is an ocean becoming a wooded area"
"An example of a chemical change is acid breaking down substances"
"An example of a fossil is a footprint in a rock"
"An example of a fossil is a paw print in rock"
"An example of a fossil is the bones of an extinct animal"
"An example of a mixture is clay mixed together"
"An example of a reproductive behavior is salmon returning to their birthplace to lay their eggs"
"""

filtered_words, words = normalize(sample)

# stemmer, lemmatizer, POS tags, NER
ps = PorterStemmer()
ps_stemmed = [ps.stem(word) for word in filtered_words]
# print(ps_stemmed)
lemmatizer = WordNetLemmatizer()
lemmatized = [lemmatizer.lemmatize(w) for w in filtered_words]
# print(lemmatized)
pos_tags = nltk.pos_tag(words)
pos_df = pd.DataFrame(pos_tags, columns=['Word', 'POS tag']).T
display(pos_df)


import spacy
from spacy import displacy
from collections import Counter
ner = spacy.load('en_core_web_sm')
doc = ner(sample)
print([(X.text, X.label_) for X in doc.ents])
displacy.render(doc, style="ent", jupyter=True)
```

	0	1	2	3	4	5	6	7	8	9	...	12368	12369	12370	12371	12372	12373	12374	12375	1
Word	a	bee	is	a	pollinating	animal	a	bird	is	a	...	transports	materials	through	the	plant	young	amphibians	breathe	thr
POS tag	DT	NN	VBZ	DT	VBG	RP	DT	NN	VBZ	DT	...	NNS	NNS	IN	DT	NN	JJ	NNS	VBP	

2 rows × 12378 columns

```
[('Earth', 'LOC')]
```

"A bee is a pollinating animal"

"A bird is a pollinating animal"

"An electrical conductor is a vehicle for the flow of electricity"

"An example of a change in the Earth LOC is an ocean becoming a wooded area"

"An example of a chemical change is acid breaking down substances"

"An example of a fossil is a footprint in a rock"

"An example of a fossil is a paw print in rock"

"An example of a fossil is the bones of an extinct animal"

"An example of a mixture is clay mixed together"

"An example of a reproductive behavior is salmon returning to their birthplace to lay their eggs"

In [14]:

```
from nltk import WordNetLemmatizer
import pandas as pd

# random sample

random_sample = "a cavern is formed by carbonic acid in groundwater seeping through rock and dissolving limest

filtered_words, words = normalize(random_sample)

# stemmer, lemmatizer, POS tags, NER
ps = PorterStemmer()
ps_stemmed = [ps.stem(word) for word in filtered_words]
# print(ps_stemmed)
lemmatizer = WordNetLemmatizer()
lemmatized = [lemmatizer.lemmatize(w) for w in filtered_words]
# print(lemmatized)
pos_tags = nltk.pos_tag(words)
pos_df = pd.DataFrame(pos_tags, columns=['Word', 'POS tag']).T
display(pos_df)


import spacy
from spacy import displacy
from collections import Counter
ner = spacy.load('en_core_web_sm')
doc = ner(sample)
print([(X.text, X.label_) for X in doc.ents])
displacy.render(doc, style="ent", jupyter=True)
```

	0	1	2	3	4	5	6	7	8	9	...	12368	12369	12370	12371	12372	12373	12374	12375	1
Word	a	bee	is	a	pollinating	animal	a	bird	is	a	...	transports	materials	through	the	plant	young	amphibians	breathe	thi
POS tag	DT	NN	VBZ	DT	VBG	RP	DT	NN	VBZ	DT	...	NNS	NNS	IN	DT	NN	JJ	NNS	VBP	

2 rows × 12378 columns

```
[('Earth', 'LOC')]
```

"A bee is a pollinating animal"

"A bird is a pollinating animal"

"An electrical conductor is a vehicle for the flow of electricity"

"An example of a change in the Earth LOC is an ocean becoming a wooded area"

"An example of a chemical change is acid breaking down substances"

"An example of a fossil is a footprint in a rock"

"An example of a fossil is a paw print in rock"

"An example of a fossil is the bones of an extinct animal"

"An example of a mixture is clay mixed together"

"An example of a reproductive behavior is salmon returning to their birthplace to lay their eggs"

Samples printed above of Stemmer, Lemmatizer, POS tags, and NER. The results are as expected with the NER being sparse and POS accurately assigning.

The function used in this assignment for normalization of text is as follows

```
def normalize(text):
    tokenized_sentences = [word_tokenize(t) for t in sent_tokenize(openbook)]
    words = [word for sentence in tokenized_sentences for word in sentence]
    words = [word.lower() for word in words if word.isalpha()]
    stop_words = stopwords.words('english')
    filtered_words = [w for w in words if w not in stop_words]
    return filtered_words
```

Part 3 - Feature Extraction

In [15]:

```
import numpy as np

# load openbook_txt
with open("OpenBookQA/data/OpenBookQA-V1-Sep2018/Data/Main/openbook.txt") as openbook_txt:
    openbook = openbook_txt.read()
with open("OpenBookQA/data/OpenBookQA-V1-Sep2018/Data/Main/openbook.txt") as openbook_txt:
    openbook_lines = openbook_txt.readlines()

def normalize(text):
    tokenized_sentences = [word_tokenize(t) for t in sent_tokenize(openbook)]
    words = [word for sentence in tokenized_sentences for word in sentence]
    words = [word.lower() for word in words if word.isalpha()]
    stop_words = stopwords.words('english')
    filtered_words = [w for w in words if w not in stop_words]
    return filtered_words, words

# tokenize words
filtered_words, words = normalize(openbook)

from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

cv = CountVectorizer(min_df=0., max_df=1.)
cv_matrix = cv.fit_transform(filtered_words)
cv_matrix = cv_matrix.toarray()
vocab = cv.get_feature_names()
```

```
cv_df = pd.DataFrame(cv_matrix, columns=vocab)
display(cv_df)

vec = CountVectorizer()
vec.fit(filtered_words)
bag_of_words = vec.transform(filtered_words)
sum_words = bag_of_words.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in     vec.vocabulary_.items()]
words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
print("10 most frequent words:", words_freq[:10])

tt = TfidfTransformer(norm='l2', use_idf=True, smooth_idf=True)
tt_matrix = tt.fit_transform(cv_matrix)
tt_matrix = tt_matrix.toarray()
vocab = cv.get_feature_names()
tt_df = pd.DataFrame(np.round(tt_matrix, 2), columns=vocab)
display(tt_df)

vec = CountVectorizer()
vec.fit(filtered_words)
bag_of_words = vec.transform(filtered_words)
sum_words = bag_of_words.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in     vec.vocabulary_.items()]
words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
print("10 most frequent words:", words_freq[:10])
```

	ability	absorb	absorbed	absorbing	absorbs	access	accurately	acid	acorns	acquire	...	working	world	wrapping	xylem	year	years
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7201	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
7202	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
7203	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
7204	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
7205	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0

7206 rows × 1711 columns

```
10 most frequent words: [('water', 124), ('used', 109), ('object', 105), ('causes', 88), ('environment', 81),
('energy', 76), ('food', 74), ('source', 68), ('animal', 66), ('increases', 66)]
```

	ability	absorb	absorbed	absorbing	absorbs	access	accurately	acid	acorns	acquire	...	working	world	wrapping	xylem	year	years
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7201	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
7202	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
7203	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
7204	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
7205	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0

7206 rows × 1711 columns

```
10 most frequent words: [('water', 124), ('used', 109), ('object', 105), ('causes', 88), ('environment', 81),
('energy', 76), ('food', 74), ('source', 68), ('animal', 66), ('increases', 66)]
```

Count Vectorizer:

Vocabulary size in CV is 1711 words (matrix 1326 x 1711)

10 most frequent words: [('water', 124), ('used', 109), ('object', 105), ('causes', 88), ('environment', 81), ('energy', 76), ('food', 74), ('source', 68), ('animal', 66), ('increases', 66)]

Tfidf Transformer:

Vocabulary size in TF Transform is 1711 words (matrix 1326 x 1711)

10 most frequent words: [('water', 124), ('used', 109), ('object', 105), ('causes', 88), ('environment', 81), ('energy', 76), ('food', 74), ('source', 68), ('animal', 66), ('increases', 66)]

The Tfidf Transform gives the same results as the Count Vectorizer.