NATURAL LANGUAGE PROCESSING

CHRIS SMITH

NATURAL LANGUAGE CAPABILITIES IN PYTHON

- Using the Natural Language Toolkit (NLTK), we can:
  - Create a frequency distribution of words
  - Remove stop words
  - Lemmatize words
- Using Scikit-Learn, we can:
  - Vectorize the data
    - N-Gram
    - Term Frequency Inverse Document Frequency (TF-IDF)

ANALYSIS WAS PERFORMED ON MAINTENANCE LOGS

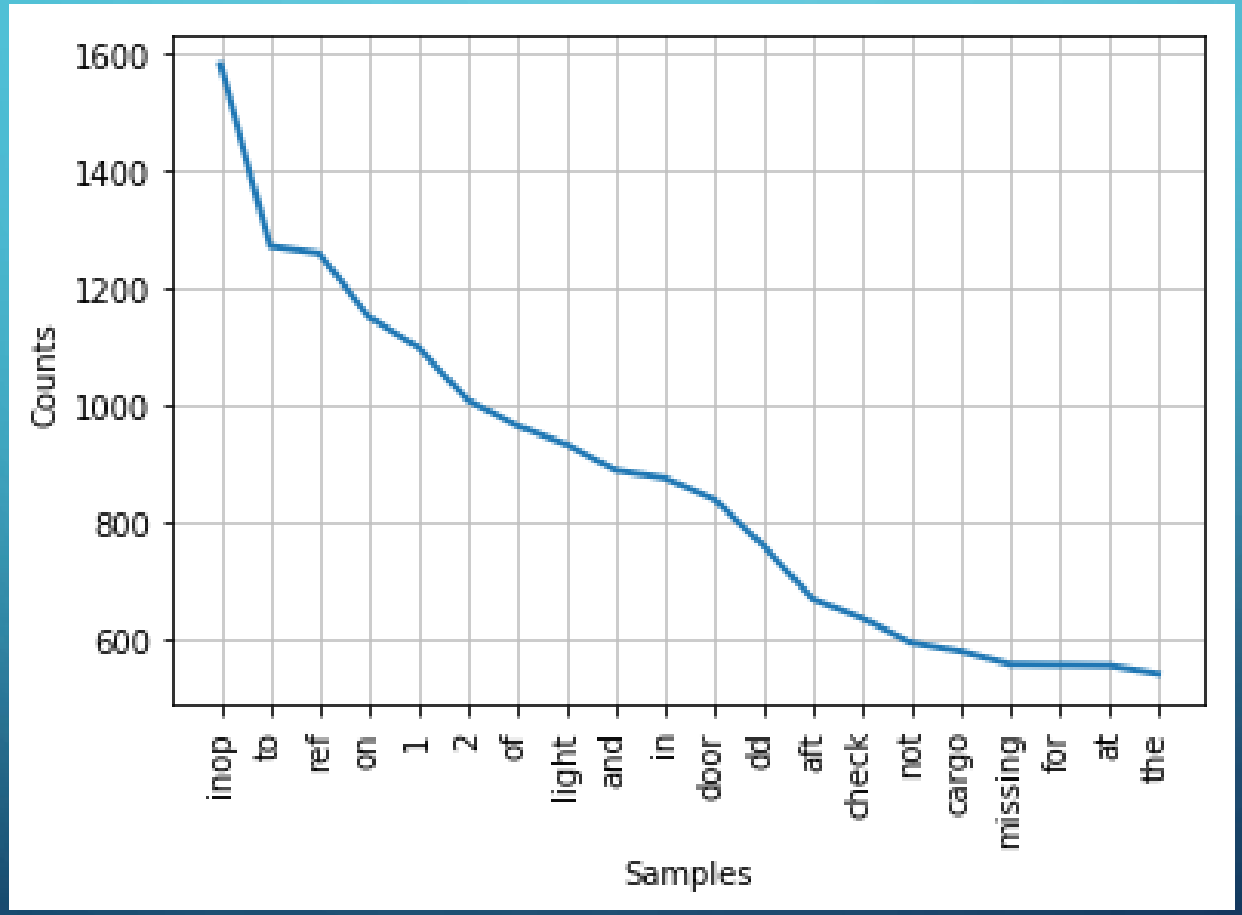
- The focus of this analysis was on airplane mechanical issues described in maintenance logs
- There are 6,860,454 rows (log entries) with 64 columns
- Analysis in this presentation will be run on the first 1,000 rows

TOKENIZATION

- Tokenization is the process of turning a string into a list of words without the punctuation. The individual words in this list are called a “tokens”.
- Example
  - Before: “Hi, my name is Chris!”
  - After: [“Hi”, “my”, “name”, “is”, “Chris”]

CAPABILITIES OF NATURAL LEARNING TOOLKIT

FREQUENCY DISTRIBUTIONS SHOW HOW MANY TIMES A WORD SHOWS UP IN THE DATA



HERE'S THE CODE TO CREATE A FREQUENCY DISTRIBUTION USING NLTK

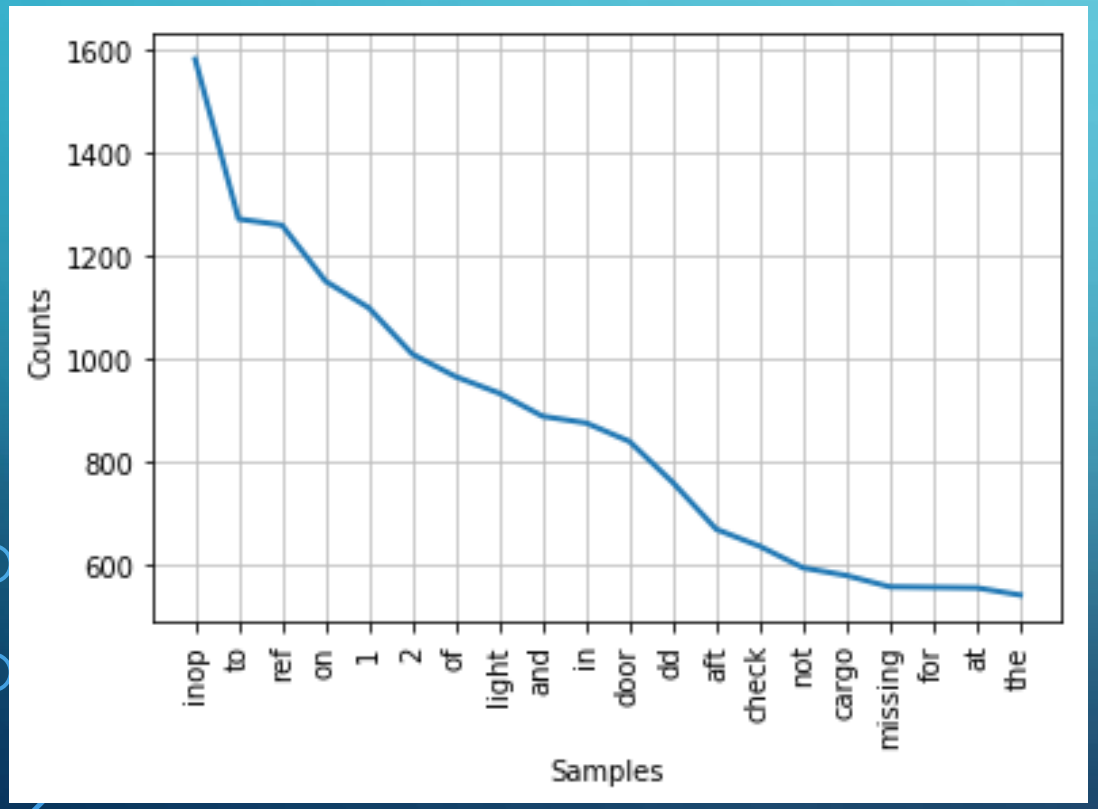
```
#Import packages
import nltk
import pandas as pd
import string

#Import data
master=pd.read_csv("filepath")

#Turn column of sentences into one long string
words= ' '.join(master["LP_DESCR_TEXT"])

#Tokenize the string
tokens = [t.lower() for t in words.split() if t not in string.punctuation]

#Create frequency distribution
freq = nltk.FreqDist(tokens)
freq.plot(20, cumulative=False)
```

NLTK CAN REMOVE STOP WORDS

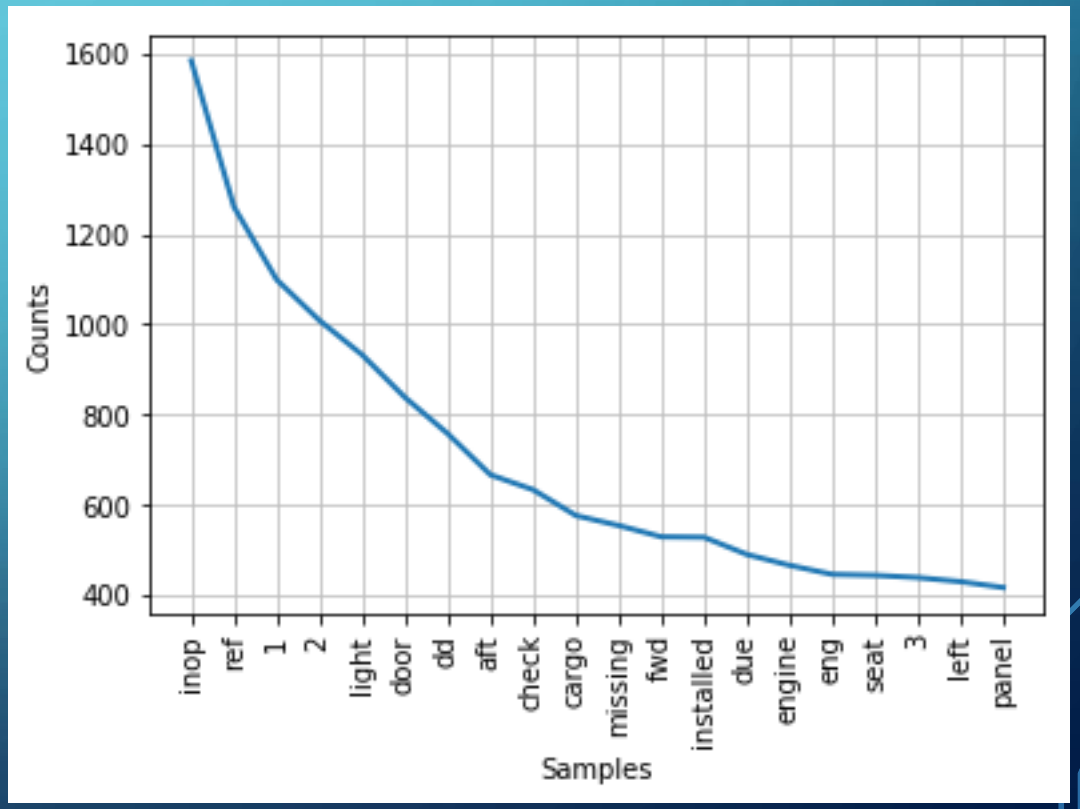
- Stop words are words that have very little meaning
- Example of removing stop words
  - Before: [“My”, “name, “is”, “Chris”]
  - After: [“My”, “name”, “Chris”]

HERE'S THE FREQUENCY DISTRIBUTION OF OUR DATA WITH STOP WORDS REMOVED

With stop words



Without stop words



HERE'S THE CODE TO REMOVE STOP WORDS

```
#Import packages
import nltk
import pandas as pd
import string
from nltk.corpus import stopwords


#Import data
master=pd.read_csv("filepath")
```

```
#Turn column of sentences into one long string
words= ' '.join(master["LP_DESCR_TEXT"])


#Tokenize the string
tokens = [t.lower() for t in words.split() if t not in string.punctuation]
```

HERE'S THE CODE TO REMOVE STOP WORDS

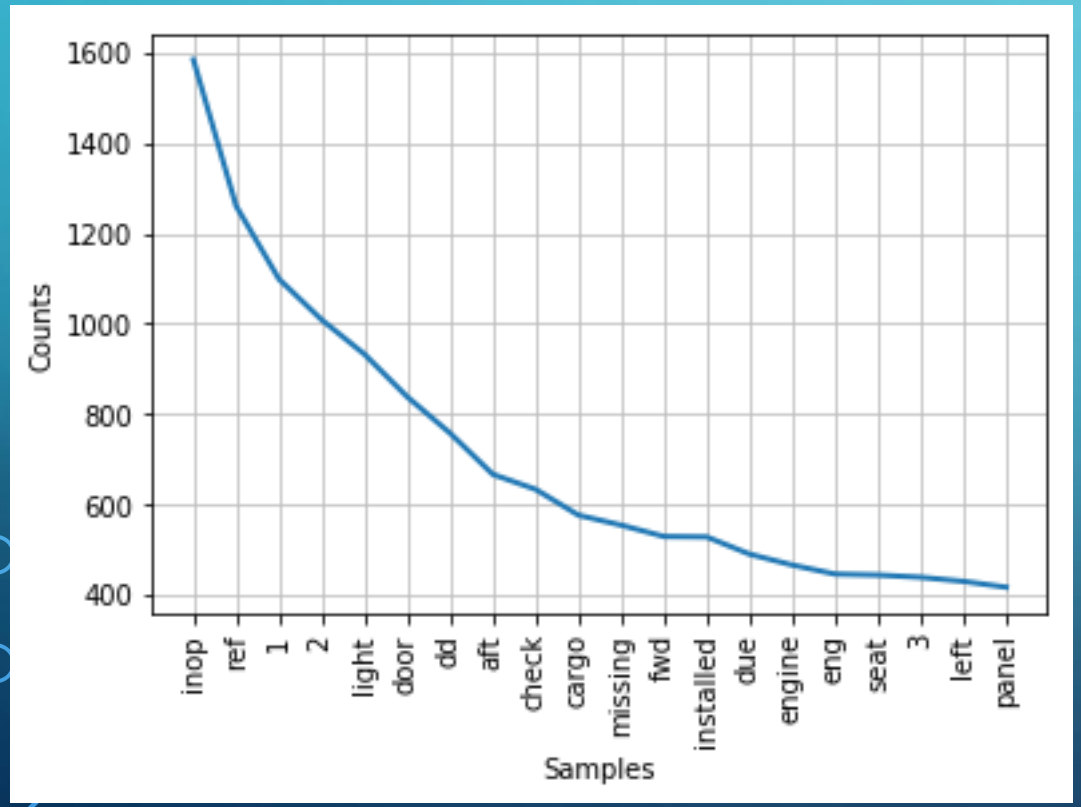
```
#Define stopwords
stopwords = stopwords.words('english')

#Remove stopwords
tokens = [t for t in tokens if t not in stopwords]

#Create frequency distribution
freq = nltk.FreqDist(tokens)
freq.plot(20, cumulative=False)
```

NLTK CAN PERFORM LEMMATIZATION

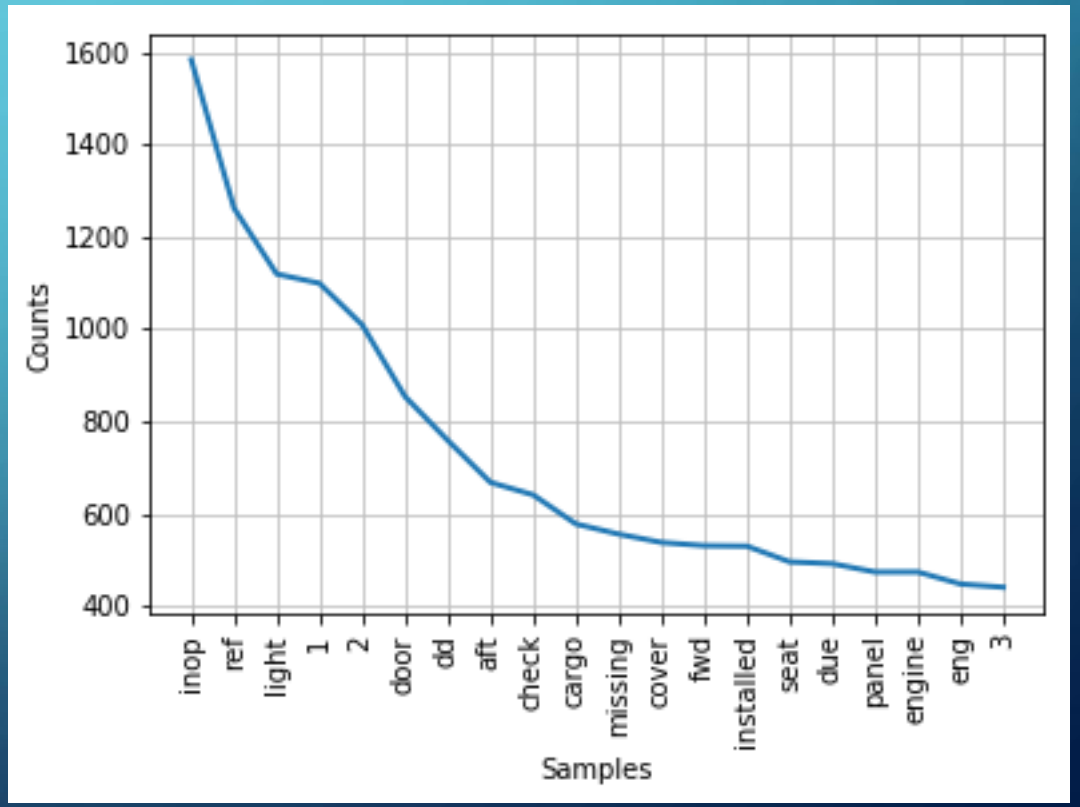
- Lemmatization is the process of grouping similar words
- Example
  - Car, cars, car's, and cars' are all similar
  - Lemmatization would group these together as "car"
  - In a frequency distribution, the number of times car, cars, car's, and cars' occurred would all be counted as "car"

HERE'S THE FREQUENCY DISTRIBUTION OF OUR DATA WITH STOP WORDS REMOVED AND WITH LEMMATIZATION

Without lemmatization



With lemmatization



HERE'S THE CODE TO REMOVE LEMMATIZE WORDS

```
#Import packages
import nltk
import pandas as pd
import string
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

#Import data
master=pd.read_csv("filepath")

#Turn column of sentences into one long string
words= ' '.join(master["LP_DESCR_TEXT"])

#Tokenize the string
tokens = [t.lower() for t in words.split() if t not in string.punctuation]
```

HERE'S THE CODE TO REMOVE LEMMATIZE WORDS

```
#Define stopwords
stopwords = stopwords.words('english')

#Set leammatizer object
wn = nltk.WordNetLemmatizer()

#Remove stopwords and lemmatize
tokens = [wn.lemmatize(t) for t in tokens if t not in stopwords]

#Create frequency distribution
freq = nltk.FreqDist(tokens)
freq.plot(20, cumulative=False)
```

CAPABILITIES OF SCIKIT-LEARN

SCIKIT-LEARN CAN BE USED TO VECTORIZE DATA

- Vectorization is the process of turning text data into numerical features
- Two types of vectorization that we will discuss:
  - NGRAM
  - Term Frequency-Inverse Document Frequency (TF-IDF)

NGRAM

- Creating a matrix where each row represents a document and the number of columns is the number of unique words in all of the documents
- If a document contains a certain word, a 1 will appear in the row corresponding to that document and the column corresponding to that word.
- Even if a word appears more than one time in a document, there will still be a 1. This is not a count, but an indicator.

NGRAM EXAMPLE

- Say we have the sentences:
  - “There used to be a Stone Age”
  - “There used to be a Bronze Age”
  - “There used to be an Iron Age”
  - “There was an Age of Revolution”
  - “Now it is a Digital Age”
- The list of unique words is: [“There”, “was”, “to”, “be”, “used”, “Stone”, “Bronze”, “Iron”, “Revolution”, “Digital”, “Age”, “of”, “Now”, “it”, “is”, “a”, “an”]

NGRAM EXAMPLE (CONT.)

- HOW “THERE USED TO BE A STONE AGE” IS VECTORIZED:

“There” = 1

“was” = 0

“to” = 1

“be” = 1

“used” = 1

“Stone” = 1

“Bronze” = 0

“Iron” = 0

“Revolution” = 0

“Digital” = 0

“Age” = 1

“of” = 0

“Now” = 0

“it” = 0

“is” = 0

“a” = 1

“an” = 0

- “There used to be a Stone Age” = [1,0,1,1,1,1,0,0,0,0,1,0,0,0,0,1,0]

NGRAM EXAMPLE (CONT.)

- How the other sentences are vectorized:
  - “There used to be a Bronze Age” = [1,0,1,1,1,0,1,0,0,0,1,0,0,0,0,1,0]
  - “There used to be an Iron Age” = [1,0,1,1,1,0,0,1,0,0,1,0,0,0,0,0,1]
  - “There was an Age of Revolution” = [1,1,0,0,0,0,0,0,1,0,1,1,0,0,0,0,1]
  - “Now it is a Digital Age” = [0,0,0,0,0,0,0,0,0,1,1,0,1,1,1,1,0]

NGRAM EXAMPLE (CONT.)

- Now, all the sentences are rows in a sparse matrix

Sentence	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	0	1	1	1	1	0	0	0	0	1	0	0	0	0	1	0
2	1	0	1	1	1	0	1	0	0	0	1	0	0	0	0	1	0
3	1	0	1	1	1	0	0	1	0	0	1	0	0	0	0	0	1
4	1	1	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1
5	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0

NGRAM EXAMPLE (CONT.)

- Scikit-Learn returns a dictionary of column indices:

- There=0
- was=1
- to=2
- be=3
- used=4
- Stone=5
- Bronze=6
- Iron=7
- Revolution=8
- Digital=9
- Age=10
- of=11
- Now=12
- it=13
- is=14
- a=15
- an=16

WHY IS IT CALLED AN NGRAM?

- The example we just discussed is a unigram because we are considering only one word at a time. For example, “There”, “used”, “to”, “be”, “a”, “Stone”, “Age”
- There are also bigrams (using two words at a time). For example, “There used”, “used to”, “to be”, “be a”, “a Stone”, and “Stone Age”.
- Using three words at a time is called a trigram. For example, “There used to”, “used to be”, “to be a”, “be a Stone”, and “a Stone Age”
- Using n words at a time is called an NGRAM

USING NGRAM ON OUR DATA

- I ran a bigram and a trigram on the first 1,000 entries in the maintenance log with lemmatization and stop words removed
- The resulting matrix was 1,000 rows (one for each entry in the maintenance log) and 7,920 columns (one for each unique set of two consecutive words or three consecutive words in the 1,000 rows)
- Most of the cells in the matrix were zero

HERE'S THE CODE TO USE NGRAM

```
#Import packages
from sklearn.feature_extraction.text import CountVectorizer


#Define cleaning function
def ngram_clean(text):
    text = "".join([word.lower() for word in text if word not in string.punctuation])
    tokens = [t for t in text.split()]
    text = " ".join([wn.lemmatize(word) for word in tokens if word not in stopwords])
    return text
```

HERE'S THE CODE TO USE NGRAM (CONT.)

```
#Apply cleaning function to text data

master[“LP_DESCR_TEXT"] = master["LP_DESCR_TEXT"].apply(lambda x: ngram_clean(x))


#Perform NGRAM on text data

ngram_vect = CountVectorizer(ngram_range=[2,3])

x_ngram_descr_text = ngram_vect.fit_transform(master[“LP_DESCR_TEXT"])


#Return dictionary of indices

ngramvocab=ngram_vect.vocabulary_
```

TERM FREQUENCY

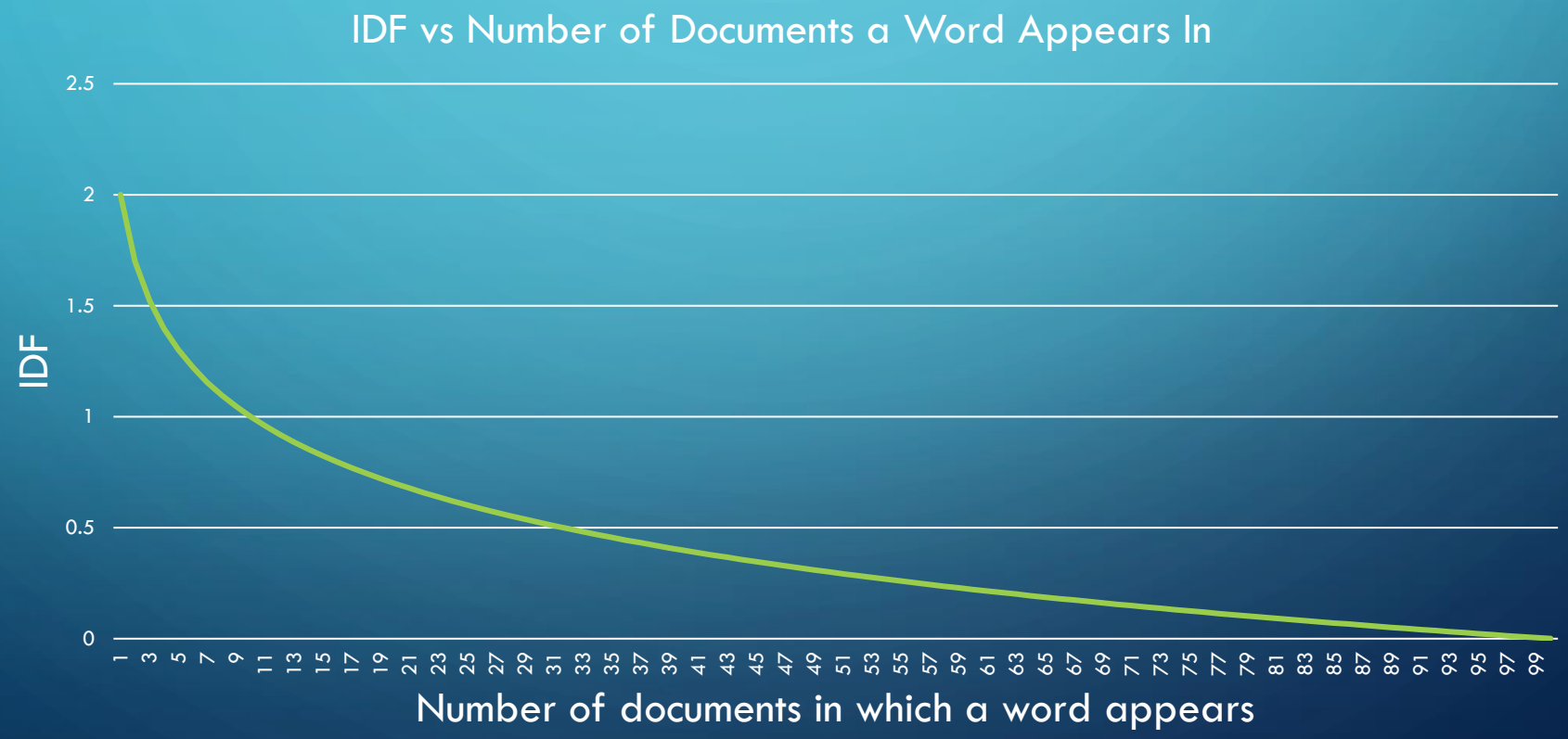
- Term Frequency $= \frac{\#\ of\ times\ a\ word\ appears\ in\ a\ document}{\#\ of\ words\ in\ a\ document}$
- Term frequency is defined as how frequently a word appears in a document
- Example
  - "I wish I could go to Canada."
  - Total words = 7
  - # of times the word "I" appears = 2
  - Term Frequency of the word "I' = 2/7 = 0.2857

INVERSE DOCUMENT FREQUENCY

- Inverse Document Frequency $= \log_{10} \frac{\#\ of\ total\ documents}{\#\ of\ documents\ in\ which\ a\ word\ appears}$
- Inverse Document Frequency measure a word's importance
- The assumption is that less frequent words are more informative and important
- Example
  - There are 10 documents, and the word "the" occurs in every document. The IDF of the word "the" is $\log_{10} \frac{10}{10} = \log_{10} 1 = 0$
  - Conversely, say the word "unicorn" is used in 1 of the 10 documents. The IDF of the word "unicorn" is $\log_{10} \frac{10}{1} = \log_{10} 10 = 1$
  - IDF considers "unicorn" to be more important than "the"

THE MORE DOCUMENTS A WORD APPEARS IN, THE LOWER THE IDF

$$\log_{10} \frac{\#\ of\ documents}{\#\ of\ documents\ in\ which\ a\ word\ appears}$$



TO FIND TF-IDF, MULTIPLY TF BY IDF

TF Table

	The	TFIDF	Process	Is	Beautiful
Document 1	.15	.04	.08	.20	.04
Document 2	.17	0	.03	.13	0
Document 3	.13	.08	0	.15	0

IDF Table

	The	TFIDF	Process	Is	Beautiful
IDF	0	0.1761	0.1761	0	0.4771

TF-IDF Table

	The	TFIDF	Process	Is	Beautiful
Document 1	0	.007	.014	0	.019
Document 2	0	0	.005	0	0
Document 3	0	.014	0	0	0

USING TF-IDF ON OUR DATA

- I used TF-IDF to vectorize the first 1,000 entries in the maintenance log with lemmatization and stop words removed
- The resulting matrix was 1,000 rows (one for each entry in the maintenance log) and 1,413 columns (one for each unique word in the 1,000 rows)
- Most of the cells in the matrix were zero

HERE ARE SOME OF THE COLUMN INDICES FROM VECTORIZING OUR DATA USING TF-IDF

- 00055=0
- ballast=86
- battery=87
- bulb=95
- cockpit=115
- coffee=117
- deferred=141
- defferred=143
- leak=192
- loose=204
- wire=344

HERE'S THE CODE TO USE NGRAM

```
#Import packages
from sklearn.feature_extraction.text import TfidfVectorizer


#Define cleaning function
def tfidf_clean(text):
    text = "".join([word.lower() for word in text if word not in string.punctuation])
    tokens = [t for t in text.split()]
    text = [wn.lemmatize(word) for word in tokens if word not in stopwords]
    return text
```

HERE'S THE CODE TO USE NGRAM (CONT.)

```
#Apply cleaning functions to text data
master[“LP_DESCR_TEXT"] = master["LP_DESCR_TEXT"].apply(lambda x: tfidf_clean(x))

#Perform NGRAM and TF-IDF on text data
tfidf_vect = TfidfVectorizer(analyzer=tfidf_clean)
x_tfidf_descr_text = tfidf_vect.fit_transform(master[“LP_DESCR_TEXT"])

#Return dictionary of indices
tfidfvocab=tfidf_vect.vocabulary_
```

WHAT ARE THE NEXT STEPS?

- Consider sentiment analysis
- Deal with misspellings
- Identify a target outcome to predict
- Use vectorized data to predict target outcome