Kyle Dalope

May 22, 2023

ADS 509-01-SU23

ADS 509 Assignment 2.1: Tokenization, Normalization, Descriptive Statistics

This notebook holds Assignment 2.1 for Module 2 in ADS 509, Applied Text Mining. Work through this notebook, writing code and answering questions where required.

In the previous assignment you put together Twitter data and lyrics data on two artists. In this assignment we explore some of the textual features of those data sets. If, for some reason, you did not complete that previous assignment, data to use for this assignment can be found in the assignment materials section of Blackboard.

This assignment asks you to write a short function to calculate some descriptive statistics on a piece of text. Then you are asked to find some interesting and unique statistics on your corpora.

General Assignment Instructions

These instructions are included in every assignment, to remind you of the coding standards for the class. Feel free to delete this cell after reading it.

One sign of mature code is conforming to a style guide. We recommend the Google Python Style Guide. If you use a different style guide, please include a cell with a link.

Your code should be relatively easy-to-read, sensibly commented, and clean. Writing code is a messy process, so please be sure to edit your final submission. Remove any cells that are not needed or parts of cells that contain unnecessary code. Remove inessential `import` statements and make sure that all such statements are moved into the designated cell.

Make use of non-code cells for written commentary. These cells should be grammatical and clearly written. In some of these cells you will have questions to answer. The questions will be marked by a "Q:" and will have a corresponding "A:" spot for you. *Make sure to answer every question marked with a* `Q:` *for full credit.*

In [ ]:
```python
import os
import re
import emoji
import pandas as pd
import numpy as np

from collections import Counter, defaultdict
from nltk.corpus import stopwords
from string import punctuation

sw = stopwords.words("english")
```

In [ ]:
```python
# Add any additional import statements you need here

import nltk
#nltk.download() #Failed attempt - [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local

import ssl
#ssl._create_default_https_context = ssl._create_unverified_context
```

In [ ]:
```python
# change `data_location` to the location of the folder on your machine.
data_location = "/Users/kyledalope/Downloads/M1 Results"

# These subfolders should still work if you correctly stored the
# data from the Module 1 assignment
```

```
twitter_folder = "twitter/"
lyrics_folder = "lyrics/"
```

In [ ]:
```
def descriptive_stats(tokens, num_tokens = 5, verbose=True) :
    """
        Given a list of tokens, print number of tokens, number of unique tokens,
        number of characters, lexical diversity (https://en.wikipedia.org/wiki/Lexical_diversity),
        and num_tokens most common tokens. Return a list with the number of tokens, number
        of unique tokens, lexical diversity, and number of characters.

    """

    # Fill in the correct values here.
    num_tokens = len(tokens) #length of tokens
    num_unique_tokens = (len(set(tokens)))
    lexical_diversity = (num_unique_tokens / num_tokens) #ratio of different unique word stems (types) to the tota
    num_characters = sum(len(token) for token in tokens) #length of the string of tokens

    if verbose :
        print(f"There are {num_tokens} tokens in the data.")
        print(f"There are {num_unique_tokens} unique tokens in the data.")
        print(f"There are {num_characters} characters in the data.")
        print(f"The lexical diversity is {lexical_diversity:.3f} in the data.")

        # print the five most common tokens
        print(Counter(tokens).most_common(5)) #p. 16 of textbook

    return([num_tokens, num_unique_tokens,
            lexical_diversity,
            num_characters])
```

In [ ]:
```
text = """here is some example text with other example text here in this text""".split()
assert(descriptive_stats(text, verbose=True)[0] == 13)
assert(descriptive_stats(text, verbose=False)[1] == 9)
assert(abs(descriptive_stats(text, verbose=False)[2] - 0.69) < 0.02)
assert(descriptive_stats(text, verbose=False)[3] == 55)
```

```
There are 13 tokens in the data.
There are 9 unique tokens in the data.
There are 55 characters in the data.
The lexical diversity is 0.692 in the data.
[('text', 3), ('here', 2), ('example', 2), ('is', 1), ('some', 1)]
```

Q: Why is it beneficial to use assertion statements in your code?

A: It is beneficial to use assertion statements in your code because they "allow you test the corectness of your code by checking if some specific conditions remain true" which can helpful for decoding purposes. If there is not a bug present in your program, then the assertion condition should always result as true. In summary, assertions are mainly for debugging and provide as checks and stops while developing so there is no introduction of new bugs when adding new features and fixing other bugs in the code.

Data Input

Now read in each of the corpora. For the lyrics data, it may be convenient to store the entire contents of the file to make it easier to inspect the titles individually, as you'll do in the last part of the assignment. In the solution, I stored the lyrics data in a dictionary with two dimensions of keys: artist and song. The value was the file contents. A data frame would work equally well.

For the Twitter data, we only need the description field for this assignment. Feel free all the descriptions read it into a data structure. In the solution, I stored the descriptions as a dictionary of lists, with the key being the artist.

In [ ]:
```
# Read in the lyrics data
lyrics =  {}
lyrics_folder = '/Users/kyledalope/Downloads/M1 Results/lyrics'

#For loop to the artist subfolders
for artist_sub in os.listdir(lyrics_folder):
    artist_path = os.path.join(lyrics_folder, artist_sub)
    if os.path.isdir(artist_path):
        lyrics[artist_sub] = {}
        #For loop to the songs for each artist subfolder
        for songs in os.listdir(artist_path):
            songs_path = os.path.join(artist_path, songs)
            if os.path.isfile(songs_path):
                with open(songs_path, 'r') as file:
                    lyrics_content = file.read()
                    lyrics[artist_sub][songs] = lyrics_content
```

In [ ]:
```
# Read in the twitter data
twitter = {}
twitter_folder = '/Users/kyledalope/Downloads/M1 Results/twitter'

#For loop to the text files in the Twitter folder
for file_name in os.listdir(twitter_folder):
    if file_name.endswith('.txt') and "_followers_data" in file_name:
        file_path = os.path.join(twitter_folder, file_name)
        artist_name = file_name.split('_followers_data')[0] #uses the cher_followers_data.txt to obtain artist nam

        with open(file_path, 'r') as file:
            twitter_descriptions = file.readlines()
            twitter_descriptions = [desc.strip() for desc in twitter_descriptions]

        twitter[artist_name] = twitter_descriptions
```

Data Cleaning

Now clean and tokenize your data. Remove punctuation chacters (available in the `punctuation` object in the `string` library), split on whitespace, fold to lowercase, and remove stopwords. Store your cleaned data, which must be accessible as an interable for `descriptive_stats`, in new objects or in new columns in your data frame.

In [ ]:
```
punctuation = set(punctuation) # speeds up comparison

#sw check stopwords
```

In [ ]:
```
# create your clean twitter data here

clean_twitter = {}
for artist_name, twitter_descriptions in twitter.items():
    clean_twitter_descriptions = []
    for twitter_description in twitter_descriptions:
        #Remove punctuation
        twitter_description = ''.join([ch for ch in twitter_description if ch not in punctuation])
        #Split on whitespace and fold to lowercase
        tokens_twitter = twitter_description.lower().split()
        #Remove stopwords
        tokens_twitter = [token for token in tokens_twitter if token not in sw]
        clean_twitter_descriptions.append(tokens_twitter)

    clean_twitter[artist_name] = clean_twitter_descriptions
```

In [ ]:
```
#Troubleshoot why twitter is not reading and dictionary is empty
#print(clean_twitter.keys())
#print(twitter)
```

In [ ]:
```
#Check file path for twitter data is correct
for file_name in os.listdir(twitter_folder):
    if file_name.endswith('.txt'):
        file_path = os.path.join(twitter_folder, file_name)
        print(file_path)
```

```
/Users/kyledalope/Downloads/M1 Results/twitter/cher_followers_data.txt
/Users/kyledalope/Downloads/M1 Results/twitter/robynkonichiwa_followers_data.txt
/Users/kyledalope/Downloads/M1 Results/twitter/cher_followers.txt
/Users/kyledalope/Downloads/M1 Results/twitter/robynkonichiwa_followers.txt
```

In [ ]:
```
# create your clean lyrics data here

clean_lyrics = {}
for artist_sub, songs in lyrics.items():
    clean_songs = {}
    for song, lyrics_content in songs.items():
        #Remove punctuation
        lyrics_clean = ''.join([ch for ch in lyrics_content if ch not in punctuation])
        #Split on whitespace and fold to lowercase
        tokens_lyrics = lyrics_clean.lower().split()
        #Remove stopwords
        tokens_lyrics = [token for token in tokens_lyrics if token not in sw]
        clean_songs[song] = tokens_lyrics

    clean_lyrics[artist_sub] = clean_songs
```

Basic Descriptive Statistics

Call your `descriptive_stats` function on both your lyrics data and your twitter data and for both artists (four total calls).

In [ ]:
```
# Descriptive stats on lyrics data for cher
cher_lyrics_ds = descriptive_stats([token for song in clean_lyrics['cher'].values() for token in song])

# Descriptive stats on Twitter data for cher
cher_twitter_ds = descriptive_stats([token for description in clean_twitter['cher'] for token in description])

# Descriptive stats on lyrics data for robyn
robyn_lyrics_ds = descriptive_stats([token for song in clean_lyrics['robyn'].values() for token in song])

# Descriptive stats on Twitter data for robyn
robyn_twitter_ds = descriptive_stats([token for description in clean_twitter['robynkonichiwa'] for token in descri
```

```
There are 35916 tokens in the data.
There are 3703 unique tokens in the data.
There are 172634 characters in the data.
The lexical diversity is 0.103 in the data.
[('love', 1004), ('im', 513), ('know', 486), ('dont', 440), ('youre', 333)]
There are 42404149 tokens in the data.
There are 10667109 unique tokens in the data.
There are 266424726 characters in the data.
The lexical diversity is 0.252 in the data.
[('0', 334292), ('1', 284601), ('2', 237886), ('love', 221728), ('3', 196690)]
There are 15227 tokens in the data.
There are 2156 unique tokens in the data.
There are 73787 characters in the data.
The lexical diversity is 0.142 in the data.
[('know', 308), ('dont', 301), ('im', 299), ('love', 275), ('got', 251)]
There are 3888265 tokens in the data.
There are 1136751 unique tokens in the data.
There are 24106279 characters in the data.
The lexical diversity is 0.292 in the data.
[('0', 31800), ('1', 24115), ('2', 17675), ('music', 16049), ('3', 14372)]
```

The descriptive stats was slightly differed as I kept coming across issues with the twitter data not pulling the artists into the dictionary and I still came across with an error where robyn had to be "robynkonichiwa" for the twitter descriptive stats specifically.

In [ ]:
```
#observe list of stop words
sw
```

```
Out[ ]: ['i',
         'me',
         'my',
         'myself',
         'we',
         'our',
         'ours',
         'ourselves',
         'you',
         "you're",
         "you've",
         "you'll",
         "you'd",
         'your',
         'yours',
         'yourself',
         'yourselves',
         'he',
         'him',
         'his',
         'himself',
         'she',
         "she's",
         'her',
         'hers',
         'herself',
         'it',
         "it's",
         'its',
         'itself',
         'they',
         'them',
         'their',
         'theirs',
         'themselves',
         'what',
         'which',
         'who',
         'whom',
         'this',
         'that',
         "that'll",
         'these',
         'those',
         'am',
         'is',
         'are',
         'was',
         'were',
         'be',
         'been',
         'being',
         'have',
         'has',
         'had',
         'having',
         'do',
         'does',
         'did',
         'doing',
         'a',
         'an',
         'the',
         'and',
         'but',
         'if',
         'or',
         'because',
         'as',
         'until',
         'while',
         'of',
         'at',
         'by',
         'for',
         'with',
         'about',
         'against',
         'between',
         'into',
         'through',
         'during',
         'before',
         'after',
         'above',
         'below',
         'to',
         'from',
         'up',
         'down',
         'in',
         'out',
         'on',
         'off',
         'over',
         'under',
         'again',
         'further',
         'then',
         'once',
         'here',
         'there',
         'when',
         'where',
         'why',
         'how',
         'all',
         'any',
         'both',
         'each',
         'few',
         'more',
         'most',
         'other',
         'some',
         'such',
         'no',
         'nor',
         'not',
         'only',
         'own',
         'same',
         'so',
         'than',
         'too',
         'very',
         's',
         't',
         'can',
         'will',
         'just',
         'don',
         "don't",
         'should',
         "should've",
         'now',
         'd',
         'll',
         'm',
         'o',
         're',
         've',
         'y',
         'ain',
         'aren',
         "aren't",
         'couldn',
         "couldn't",
         'didn',
         "didn't",
         'doesn',
         "doesn't",
         'hadn',
         "hadn't",
         'hasn',
         "hasn't",
         'haven',
         "haven't",
         'isn',
         "isn't",
         'ma',
         'mightn',
         "mightn't",
         'mustn',
         "mustn't",
         'needn',
         "needn't",
         'shan',
         "shan't",
         'shouldn',
         "shouldn't",
         'wasn',
         "wasn't",
         'weren',
         "weren't",
         'won',
         "won't",
         'wouldn',
         "wouldn't"]
```

Q: How do you think the "top 5 words" would be different if we left stopwords in the data?

A: I believe if the stopwords were left in the data, the top 5 words would have been: 'i', 'me', 'my', 'myself', and 'we'. Which would not have provided much insight when seeing the top 5 words only. In this case, I believe it was beneficial to remove the stopwords and allow a little more focus in identifying distinct words for analysis.

Q: What were your prior beliefs about the lexical diversity between the artists? Does the difference (or lack thereof) in lexical diversity between the artists conform to your prior beliefs?

A: When defining lexical diversity, it "refers to the range and variety of vocabulary deployed in a text by a speaker/writer." Thus, for Cher and Robyn I believed that Cher would have had a higher diversity and range when compared to Robyn. But after observing the descriptive stats, that is not the case which showcases slightly more lexical diversity with Robyn. However, that makes a little more sense logically as Cher had a consistent and repetitive genre and in songs.

Specialty Statistics

The descriptive statistics we have calculated are quite generic. You will now calculate a handful of statistics tailored to these data.

1. Ten most common emojis by artist in the twitter descriptions.
2. Ten most common hashtags by artist in the twitter descriptions.
3. Five most common words in song titles by artist.
4. For each artist, a histogram of song lengths (in terms of number of tokens)

We can use the `emoji` library to help us identify emojis and you have been given a function to help you.

In [ ]:
```python
assert(emoji.is_emoji("❤"))
assert(not emoji.is_emoji(":-)"))
```

Emojis 😁

What are the ten most common emojis by artist in the twitter descriptions?

In [ ]:
```python
# Ten most common emojis by artist in the twitter descriptions
```

```
emojis = defaultdict(list)

for artist in twitter:
    for desc in twitter[artist]:
        emojis[artist].extend(ch for ch in desc if emoji.is_emoji(ch))
```

In [ ]:
```
for artist in emojis:
    print(artist)
    print(Counter(emojis[artist]).most_common(10))
```

```
cher
[('❤', 94506), ('🌈', 66291), ('♥', 48059), ('🏳', 47174), ('✨', 45846), ('🌊', 31234), ('💙', 31050), ('', 251
95), ('✌', 21963), ('💜', 21571)]
robynkonichiwa
[('🌈', 6086), ('❤', 5635), ('🏳', 4641), ('♥', 4249), ('✨', 3217), ('', 1751), ('✌', 1495), ('♀', 1347), ('',
1340), ('💙', 1200)]
```

Hashtags

What are the ten most common hashtags by artist in the twitter descriptions?

In [ ]:
```
# Ten most common hashtags by artist in the twitter descriptions

hashtags = defaultdict(list)

for artist in twitter:
    for desc in twitter[artist]:

        hashtags[artist].extend([item.lower() for item in desc.split() if item.startswith('#')])
```

In [ ]:
```
for artist in hashtags:
    print(artist)
    print(Counter(hashtags[artist]).most_common(10))
```

```
cher
[('#resist', 9729), ('#blm', 9271), ('#blacklivesmatter', 7770), ('#fbr', 2991), ('#theresistance', 2929), ('#1',
2580), ('#resistance', 2438), ('#', 2205), ('#voteblue', 1939), ('#lgbtq', 1461)]
robynkonichiwa
[('#blacklivesmatter', 601), ('#blm', 365), ('#music', 262), ('#1', 213), ('#', 175), ('#teamfollowback', 118),
('#edm', 104), ('#resist', 77), ('#freebritney', 70), ('#blacktranslivesmatter', 58)]
```

Song Titles

What are the five most common words in song titles by artist? The song titles should be on the first line of the lyrics pages, so if you have kept the raw file contents around, you will not need to re-read the data.

In [ ]:
```
# Five most common words in song titles by artist

songtitle_tokens = defaultdict(list)

for artist in lyrics:
    for song, page in lyrics[artist].items():
        page = page.split("\n")

        songtitle_tokens[artist].extend([item.lower() for item in page[0].split()])
```

In [ ]:
```
for artist in songtitle_tokens:
    print(artist)
    print(Counter(songtitle_tokens[artist]).most_common(5))
```

```
robyn
[('me', 7), ('you', 7), ('the', 7), ('my', 6), ('to', 6)]
cher
[('the', 29), ('to', 28), ('"the', 24), ('of', 21), ('"i', 21)]
```

The code chunks above were observed from Office Hours.

Song Lengths

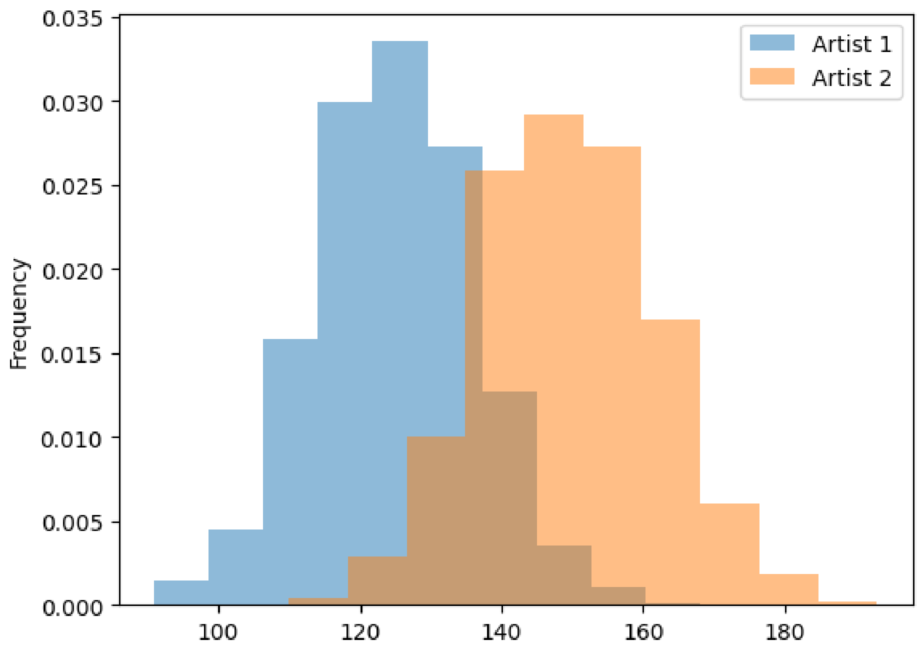
For each artist, a histogram of song lengths (in terms of number of tokens). If you put the song lengths in a data frame with an artist column, matplotlib will make the plotting quite easy. An example is given to help you out.

In [ ]:
```
num_replicates = 1000

df = pd.DataFrame({
    "artist" : ['Artist 1'] * num_replicates + ['Artist 2']*num_replicates,
    "length" : np.concatenate((np.random.poisson(125,num_replicates),np.random.poisson(150,num_replicates)))
})

df.groupby('artist')['length'].plot(kind="hist",density=True,alpha=0.5,legend=True)
```

Out[ ]:
```
artist
Artist 1    AxesSubplot(0.125,0.11;0.775x0.77)
Artist 2    AxesSubplot(0.125,0.11;0.775x0.77)
Name: length, dtype: object
```



Since the lyrics may be stored with carriage returns or tabs, it may be useful to have a function that can collapse whitespace, using regular expressions, and be used for splitting.

Q: What does the regular expression `'\s+'` match on?

A: The regular expression `'\s+'` matches on one or more consecutive white space characters, this includes space, tab, form feed, line feed, and other Unicode spaces.

In [ ]:
```
collapse_whitespace = re.compile(r'\s+')

def tokenize_lyrics(lyric) :
    """strip and split on whitespace"""
    return([item.lower() for item in collapse_whitespace.split(lyric)])
```

In [ ]:
```
# Your lyric length comparison chart here.
artist_vector = []
song_vector = []
lyrics_vector = []

for artist in lyrics:
    for song, song_lyric in lyrics[artist].items():
        artist_vector.append(artist)
        song_vector.append(song)
        lyrics_vector.append(song_lyric)

lyrics_df = pd.DataFrame()
lyrics_df['artist'] = artist_vector
lyrics_df['song'] = song_vector
lyrics_df['lyrics'] = lyrics_vector
```

```
lyrics_df.head()
```

Out[ ]:

	artist	song	lyrics
0	robyn	robyn_includemeout.txt	"Include Me Out"\n\n\n\nIt is really very simp...
1	robyn	robyn_electric.txt	"Electric"\n\n\n\nElectric...\n\nIt's electric...
2	robyn	robyn_beach2k20.txt	"Beach 2K20"\n\n\n\n(So you wanna go out?\nHow...
3	robyn	robyn_lovekills.txt	"Love Kills"\n\n\n\nIf you're looking for love...
4	robyn	robyn_timemachine.txt	"Time Machine"\n\n\n\nHey, what did I do?\nCan...

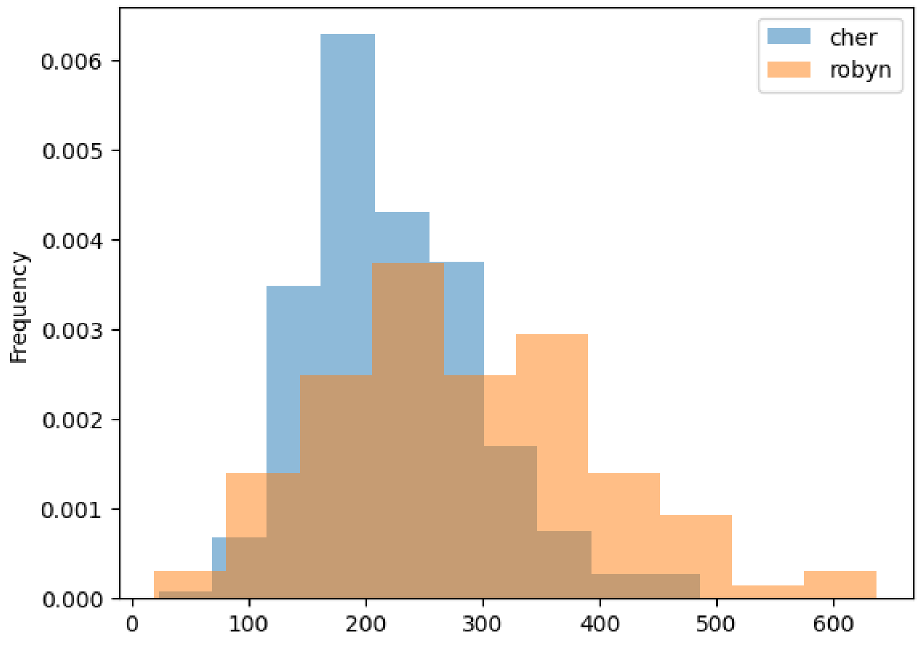
In [ ]:

```
#tokenize the lyrics_df
lyrics_df['lyric_token'] = lyrics_df['lyrics'].apply(tokenize_lyrics)
lyrics_df['lyric_len'] = lyrics_df['lyric_token'].apply(len)

#Group the artists and their lyrics then plot the histogram
lyrics_df.groupby('artist')['lyric_len'].plot(kind="hist",density=True,alpha=0.5,legend=True)
```

Out[ ]:

```
artist
cher     AxesSubplot(0.125,0.11;0.775x0.77)
robyn    AxesSubplot(0.125,0.11;0.775x0.77)
Name: lyric_len, dtype: object
```



References

YS, L. S. (n.d.). lexicalrichness: A small module to compute textual lexical richness (aka lexical diversity). PyPI. Retrieved May 23, 2023, from https://pypi.org/project/lexicalrichness/#:~:text=LexicalRichness%20is%20a%20small%20Python

Wikipedia Contributors. (2019, February 16). Lexical diversity. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Lexical_diversity

python - Best way to strip punctuation from a string. (n.d.). Stack Overflow. https://stackoverflow.com/questions/265960/best-way-to-strip-punctuation-from-a-string

Regular expression syntax cheatsheet - JavaScript | MDN. (n.d.). Developer.mozilla.org. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Cheatsheet Built-in Types. (n.d.). Python Documentation. https://docs.python.org/3/library/stdtypes.html#str.join

6. Expressions. (n.d.). Python Documentation. Retrieved May 23, 2023, from https://docs.python.org/3/reference/expressions.html#generator-expressions

Emoji: Extract, Analyze, and Get Insights — Python. (n.d.). Advertools.readthedocs.io. Retrieved May 23, 2023, from https://advertools.readthedocs.io/en/master/advertools.emoji.html

Albrecht, J., Ramachandran, S., & Winkler, C. (2020). Blueprints for text analytics using Python. O'Reilly.

NLTK :: Installing NLTK Data. (n.d.). Www.nltk.org. https://www.nltk.org/data.html

disable default certificate verification in python 2.7.9. (n.d.). Stack Overflow. Retrieved May 23, 2023, from https://stackoverflow.com/questions/30461969/disable-default-certificate-verification-in-python-2-7-9