Experiment No. 2

Title: Implementation of removal of punctuations, stop words, extra white spaces, URLs and HTML code from Text

Batch: 1 **Roll No.: 16010420008** **Experiment No.:2**

Aim: To implement removal of punctuations, stop words, extra white spaces, URLs and HTML code from Text.

Resources needed: Text Editor, Python Interpreter

Theory:

After Tokenization, the next step is NLP pipeline is stop words removal. Depending on the requirement, along with stop words, punctuations, extra white spaces, URLs and HTML code are also removed sometimes.

Remove stop words

Stop words are common words that do not add significant meaning to the text, such as “a,” “an,” and “the” and so on.

To remove common stop words from a list of tokens using NLTK, the nltk.corpus.stopwords.words() function can be used to get a list of stopwords in a specific language and filter the tokens using this list.

Example:

```
import nltk import word_tokenize
from nltk.corpus import stopwords

# input text
text = "Natural language processing is a field of artificial intelligence that
deals with the interaction between computers and human (natural) language."

# tokenize the text
tokens = word_tokenize(text)

# get list of stopwords in English
stopwords = stopwords.words("english")

# remove stopwords
filtered_tokens = [token for token in tokens if token.lower() not in
stopwords]

print("Tokens without stopwords:", filtered_tokens)
```

Output generated:

```
Tokens without stopwords: ['Natural', 'language', 'processing', 'field',
'artificial', 'intelligence', 'deals', 'interaction', 'computers', 'human',
'(', 'natural', ')', 'language', '.']
```

Note: When you run this code the first time, it is possible that you will get a Python error, including the following message at the end: Resource stopwords not found. Please use the NLTK Downloader to obtain the resource:

import nltk

nltk.download('stopwords')

Remove punctuations

Removing punctuation marks simplifies the text and make it easier to process.To remove punctuation from a list of tokens using NLTK, string module can be used to check if each token is a punctuation character.

Example:

```
import nltk
import string

# input text
text = "Natural language processing is a field of artificial intelligence that
deals with the interaction between computers and human (natural) language."

# tokenize the text
tokens = nltk.word_tokenize(text)

# remove punctuation
filtered_tokens = [token for token in tokens if token not in
string.punctuation]

print("Tokens without punctuation:", filtered_tokens)
```

Output generated:

```
Tokens without punctuation: ['Natural', 'language', 'processing', 'is', 'a',
'field', 'of', 'artificial', 'intelligence', 'that', 'deals', 'with', 'the',
'interaction', 'between', 'computers', 'and', 'human', 'natural', 'language']
```

Remove extra whitespaces

To remove extra white space from a string of text using NLTK the string.strip() function can be used to remove leading and trailing white space, and the string.replace() function can be used to replace multiple consecutive white space characters with a single space.

```
import nltk
import string

# input text with extra white space
text = "  Natural    language processing    is    a field    of artificial
intelligence    that deals with the interaction between computers and human
(natural)    language.    "

# remove leading and trailing white space
```

```
text = text.strip()

# replace multiple consecutive white space characters with a single space
text = " ".join(text.split())

print("Cleaned text:", text)
```

Output generated:

```
Cleaned text: Natural language processing is a field of artificial
intelligence that deals with the interaction between computers and human
(natural) language.
```

Remove URLs

To remove URLs from a string of text using NLTK, a regular expression pattern can be used to identify URLs and can be replaced with an empty string.

Example:

```
import nltk
import re

# input text with URLs
text = "Natural language processing is a field of artificial intelligence that
deals with the interaction between computers and human (natural) language.
Check out this article for more information:
https://en.wikipedia.org/wiki/Natural_language_processing"

# define a regular expression pattern to match URLs
pattern = r"(http|ftp|https)://([\w_-]+(?:(?:\.[\w_-]+)+))([\w.,@?^=%&:/~+#-
]*[\w@?^=%&/~+#-])?"

# replace URLs with an empty string
cleaned_text = re.sub(pattern, "", text)

print("Text without URLs:", cleaned_text)
```

Output generated:

```
Text without URLs: Natural language processing is a field of artificial
intelligence that deals with the interaction between computers and human
(natural) language. Check out this article for more information:
```

Remove HTML Code

To remove HTML code from a string of text using NLTK, regular expression pattern can be used to identify HTML tags and then can be replaced with an empty string.

Example:

```
import nltk
import re

# input text with HTML code
text = "Natural language processing is a field of artificial intelligence that
deals with the interaction between computers and human (natural) language.
<b>This is an example of bold text.</b>"

# define a regular expression pattern to match HTML tags
pattern = r"<[^>]+>"

# replace HTML tags with an empty string
cleaned_text = re.sub(pattern, "", text)

print("Text without HTML code:", cleaned_text)
```

Output generated:

Text without HTML code: Natural language processing is a field of artificial intelligence that deals with the interaction between computers and human (natural) language. This is an example of bold text.

Note that sometimes depending on the need, for example, while working with text data derived from speech, punctuations might be required to be added to the extracted text. Similarly, rather than removing URLs and HTML code from the text, this data might be of interest to us while working with some NLP applications.

Activity:

1. Add custom list of stop words to English language stop words and use this list of stop words to remove stop words from text
2. Apply stop word removal, punctuation removal, space removal, URL and HTML code removal to a dataset of technical discussion forum such as dataset of stack overflow.

Results: (Program with snapshot of output)

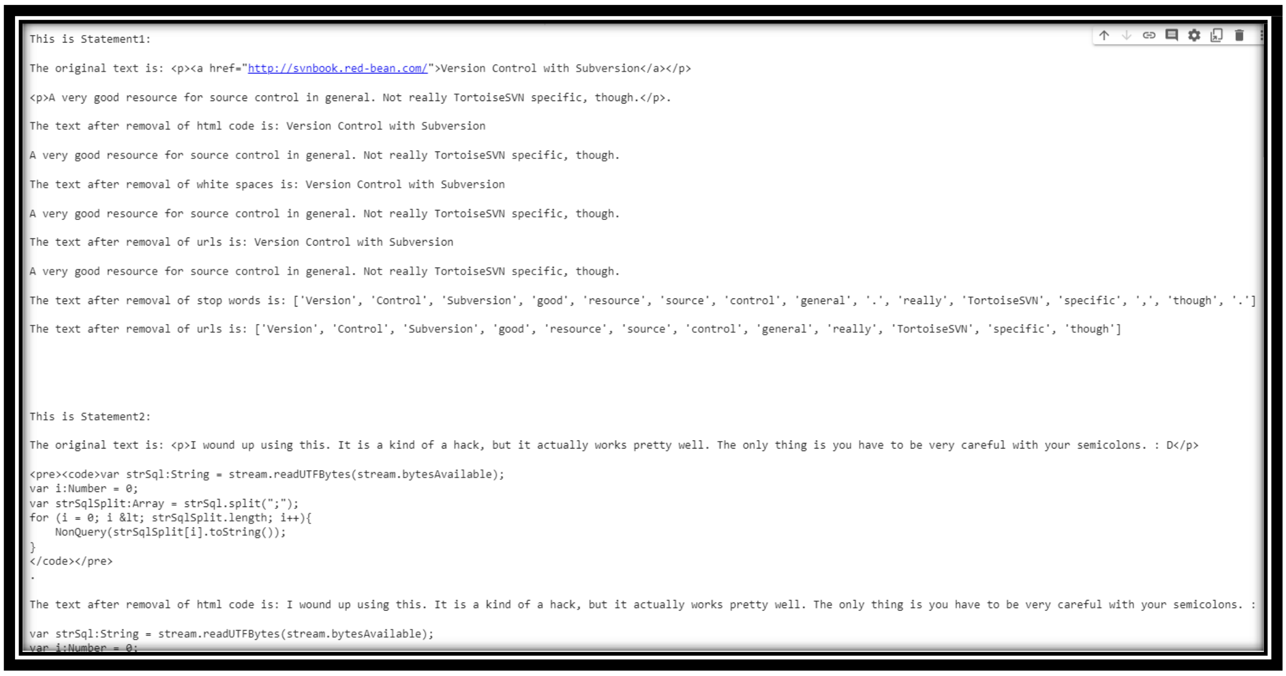
Code:

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
import string
import re
nltk.download('punkt')
nltk.download('stopwords')
from nltk import word_tokenize
```

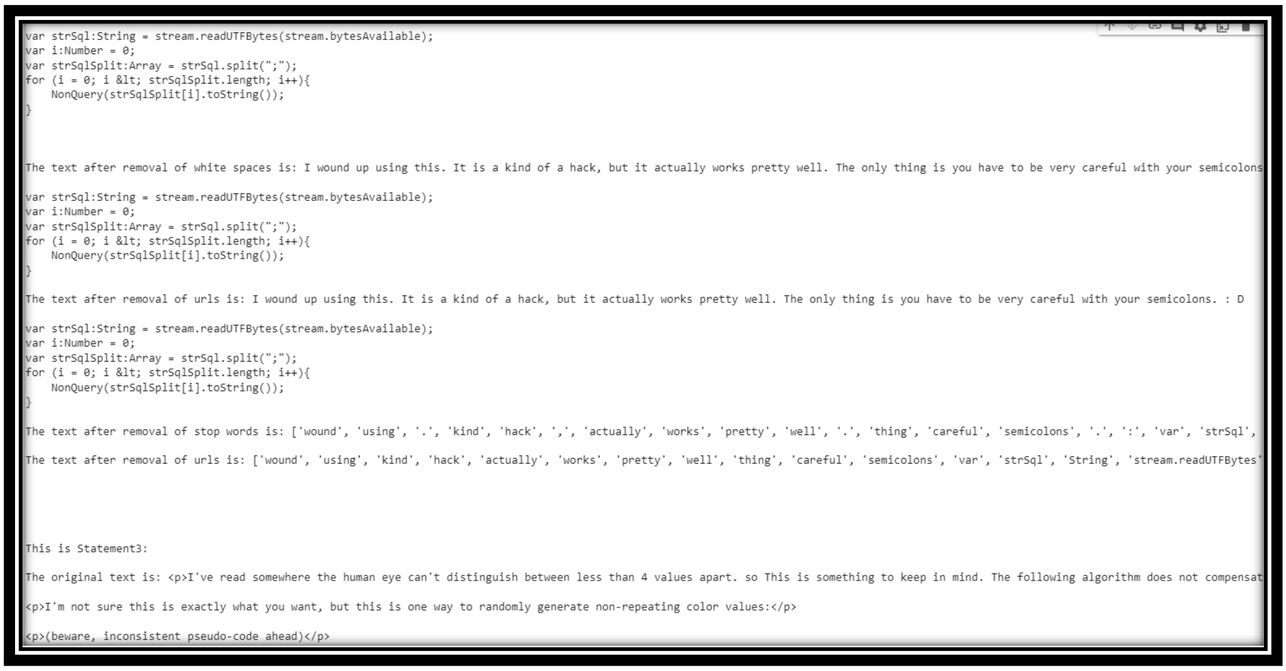
```
  def stopwords_removal(stopwords, token_words):
      stopwords = stopwords.words("english")
      stopwords.append("ies")
      stopwords.append("ed")
      stopwords_removal_tokens = [token for token in token_words if token.lower()
not in stopwords]
      return stopwords_removal_tokens

  def punctuations_removal(token_words):
      punctuations_removal_tokens = [token for token in token_words if token not
in string.punctuation]
      return punctuations_removal_tokens

  def whitespace_removal(text):
      text = text.strip()
      whitespace_removal_text = " ".join(text.split())
      return text

  def url_removal(text):
      pattern = r"(http|ftp|https)://([\w_-]+(?:(?:\.[\w_-]+)+))([\w.,@?^=%&:/~+#-
]*[\w@?^=%&/~+#-])?"
      url_removal_text = re.sub(pattern, "", text)
      return url_removal_text

  def html_removal(text):
      pattern = r"<[^>]+>"
      html_removal_text = re.sub(pattern, "", text)
      return html_removal_text

  csv_file = pd.read_csv('/content/Answers.csv')
  csv_file_column = csv_file[csv_file['Body'].notna()]

  answer_column = []
  for i in range(0, 10):
      answer_column.append(csv_file_column['Body'][i])

  for i in answer_column:
      print(f"This is Statement{answer_column.index(i) + 1}: \n")
      print(f"The original text is: {i}.\n")
      html_removal_text = html_removal(i)
      print(f"The text after removal of html code is: {html_removal_text}\n")
      whitespace_removal_text = whitespace_removal(html_removal_text)
```

```
    print(f"The text after removal of white spaces is:
{whitespace_removal_text}\n")
    url_removal_text = url_removal(whitespace_removal_text)
    print(f"The text after removal of urls is: {url_removal_text}\n")
    tokens = word_tokenize(url_removal_text)
    stopwords_removal_tokens = stopwords_removal(stopwords, tokens)
    print(f"The text after removal of stop words is:
{stopwords_removal_tokens}\n")
    punctuations_removal_tokens =
punctuations_removal(stopwords_removal_tokens)
    print(f"The text after removal of urls is:
{punctuations_removal_tokens}\n\n\n\n\n\n")
```

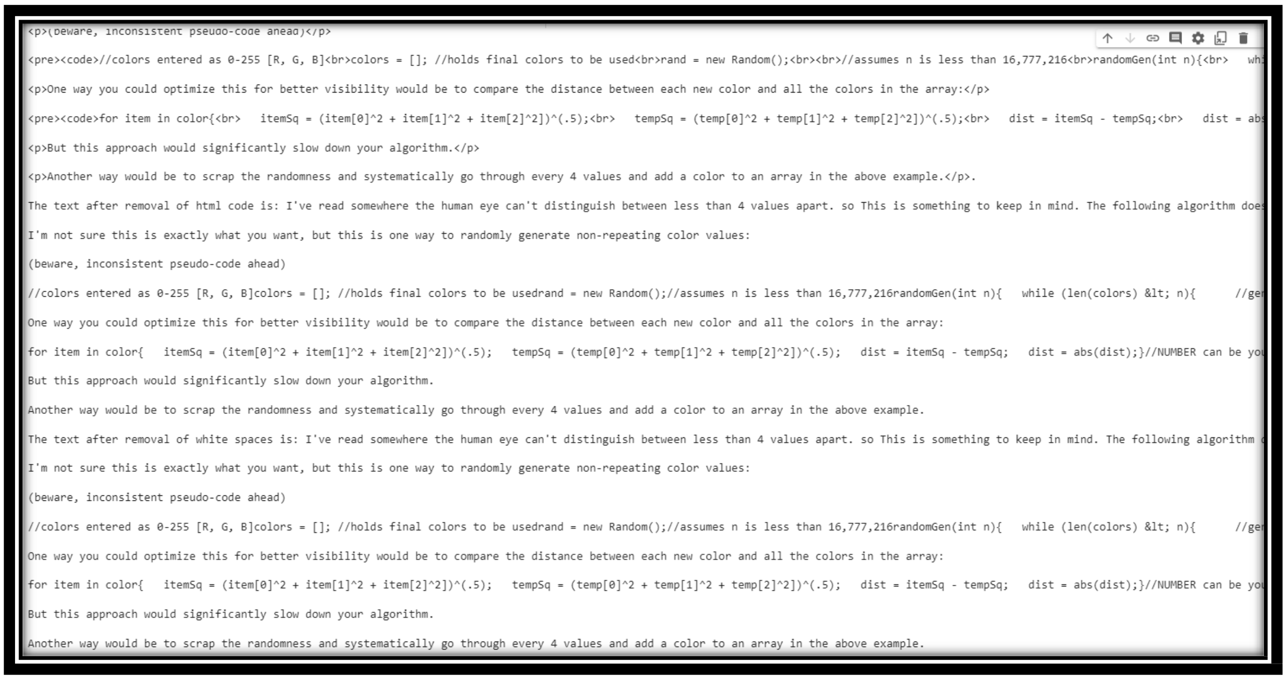
Output:



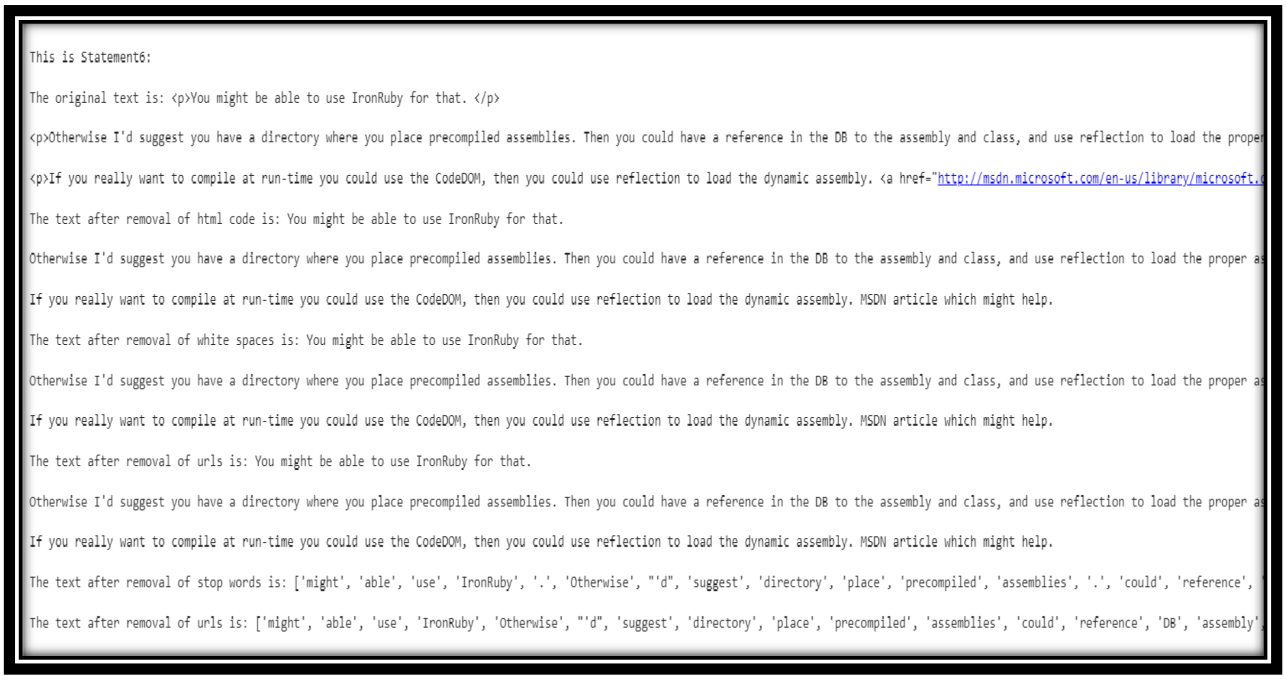
```
This is Statement1:

The original text is: <p><a href="http://svnbook.red-bean.com/">Version Control with Subversion</a></p>

<p>A very good resource for source control in general. Not really TortoiseSVN specific, though.</p>.

The text after removal of html code is: Version Control with Subversion

A very good resource for source control in general. Not really TortoiseSVN specific, though.

The text after removal of white spaces is: Version Control with Subversion

A very good resource for source control in general. Not really TortoiseSVN specific, though.

The text after removal of urls is: Version Control with Subversion

A very good resource for source control in general. Not really TortoiseSVN specific, though.

The text after removal of stop words is: ['Version', 'Control', 'Subversion', 'good', 'resource', 'source', 'control', 'general', '.', 'really', 'TortoiseSVN', 'specific', ',', 'though', '.']

The text after removal of urls is: ['Version', 'Control', 'Subversion', 'good', 'resource', 'source', 'control', 'general', 'really', 'TortoiseSVN', 'specific', 'though']




This is Statement2:

The original text is: <p>I wound up using this. It is a kind of a hack, but it actually works pretty well. The only thing is you have to be very careful with your semicolons. : D</p>

<pre><code>var strSql:String = stream.readUTFBytes(stream.bytesAvailable);
var i:Number = 0;
var strSqlSplit:Array = strSql.split(";");
for (i = 0; i &lt; strSqlSplit.length; i++){
    NonQuery(strSqlSplit[i].toString());
}
</code></pre>
.

The text after removal of html code is: I wound up using this. It is a kind of a hack, but it actually works pretty well. The only thing is you have to be very careful with your semicolons. :

var strSql:String = stream.readUTFBytes(stream.bytesAvailable);
var i:Number = 0;
```



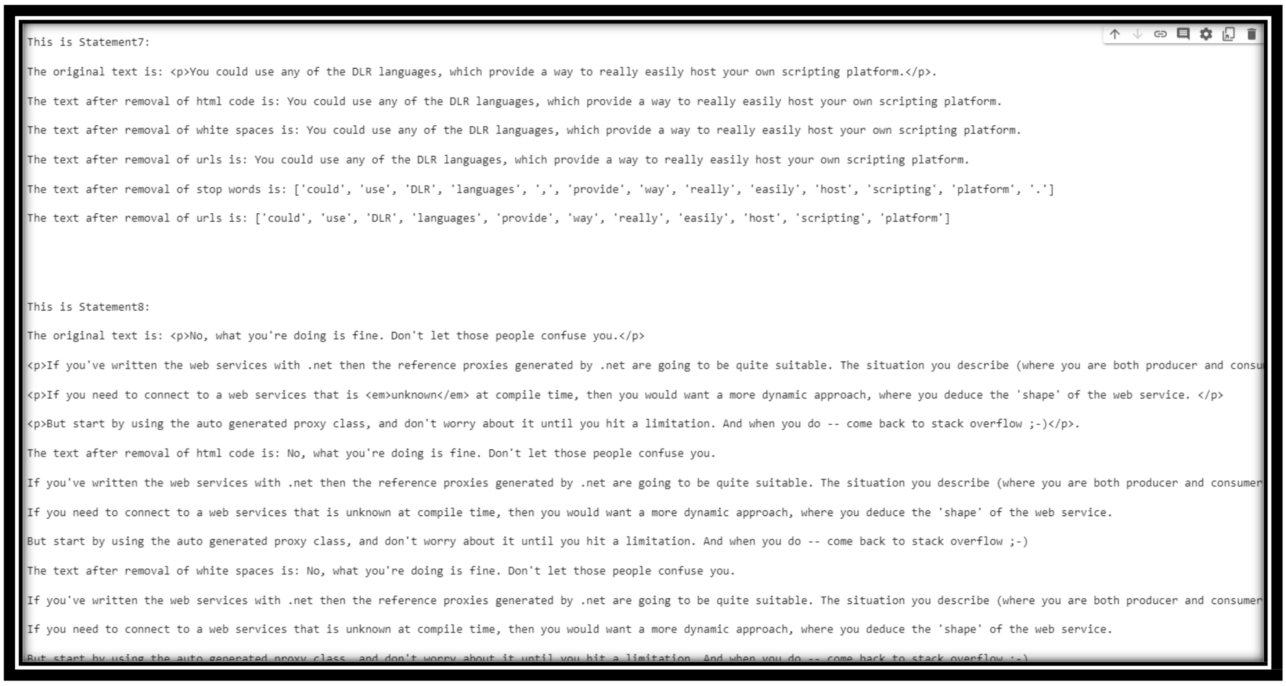
```
var strSql:String = stream.readUTFBytes(stream.bytesAvailable);
var i:Number = 0;
var strSqlSplit:Array = strSql.split(";");
for (i = 0; i &lt; strSqlSplit.length; i++){
    NonQuery(strSqlSplit[i].toString());
}



The text after removal of white spaces is: I wound up using this. It is a kind of a hack, but it actually works pretty well. The only thing is you have to be very careful with your semicolons

var strSql:String = stream.readUTFBytes(stream.bytesAvailable);
var i:Number = 0;
var strSqlSplit:Array = strSql.split(";");
for (i = 0; i &lt; strSqlSplit.length; i++){
    NonQuery(strSqlSplit[i].toString());
}

The text after removal of urls is: I wound up using this. It is a kind of a hack, but it actually works pretty well. The only thing is you have to be very careful with your semicolons. : D

var strSql:String = stream.readUTFBytes(stream.bytesAvailable);
var i:Number = 0;
var strSqlSplit:Array = strSql.split(";");
for (i = 0; i &lt; strSqlSplit.length; i++){
    NonQuery(strSqlSplit[i].toString());
}

The text after removal of stop words is: ['wound', 'using', '.', 'kind', 'hack', ',', 'actually', 'works', 'pretty', 'well', '.', 'thing', 'careful', 'semicolons', '.', ':', 'var', 'strSql',

The text after removal of urls is: ['wound', 'using', 'kind', 'hack', 'actually', 'works', 'pretty', 'well', 'thing', 'careful', 'semicolons', 'var', 'strSql', 'String', 'stream.readUTFBytes'




This is Statement3:

The original text is: <p>I've read somewhere the human eye can't distinguish between less than 4 values apart. so This is something to keep in mind. The following algorithm does not compensat

<p>I'm not sure this is exactly what you want, but this is one way to randomly generate non-repeating color values:</p>

<p>(beware, inconsistent pseudo-code ahead)</p>
```



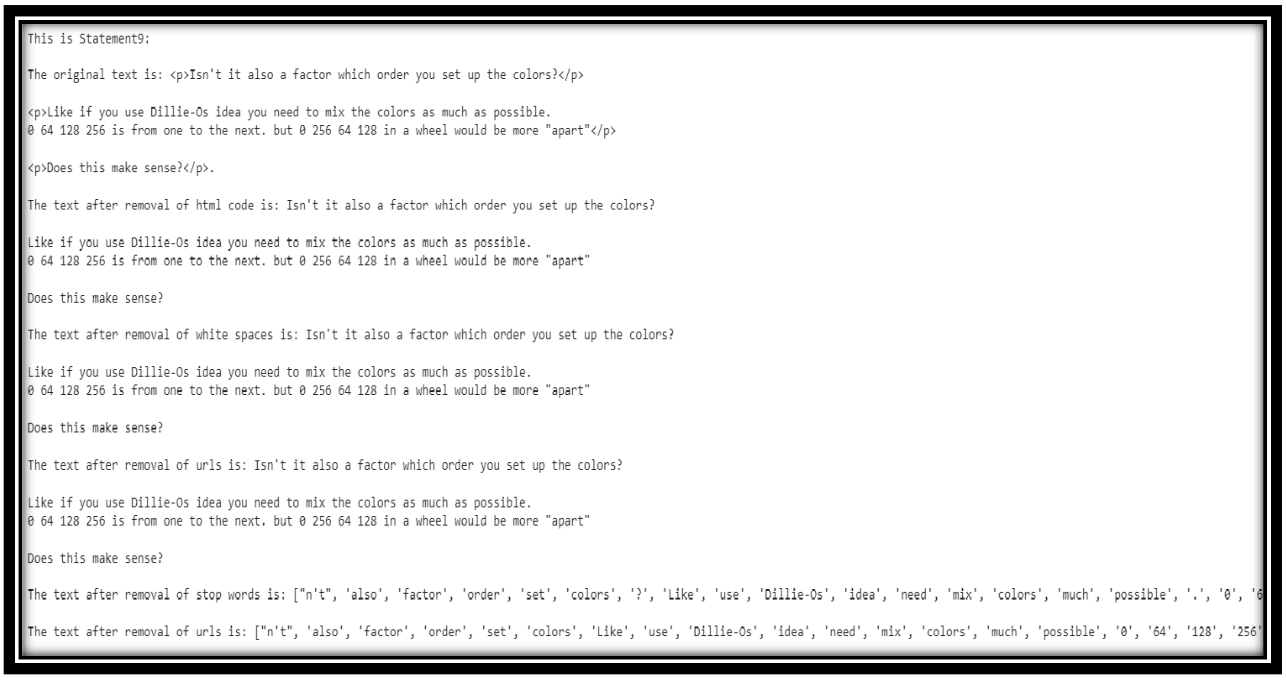
```
<p>(beware, inconsistent pseudo-code ahead)</p>

<pre><code>//colors entered as 0-255 [R, G, B]<br>colors = []; //holds final colors to be used<br>rand = new Random();<br><br>//assumes n is less than 16,777,216<br>randomGen(int n){<br>   wh

<p>One way you could optimize this for better visibility would be to compare the distance between each new color and all the colors in the array:</p>

<pre><code>for item in color{<br>   itemSq = (item[0]^2 + item[1]^2 + item[2]^2])^(.5);<br>   tempSq = (temp[0]^2 + temp[1]^2 + temp[2]^2])^(.5);<br>   dist = itemSq - tempSq;<br>   dist = abs

<p>But this approach would significantly slow down your algorithm.</p>

<p>Another way would be to scrap the randomness and systematically go through every 4 values and add a color to an array in the above example.</p>.

The text after removal of html code is: I've read somewhere the human eye can't distinguish between less than 4 values apart. so This is something to keep in mind. The following algorithm does

I'm not sure this is exactly what you want, but this is one way to randomly generate non-repeating color values:

(beware, inconsistent pseudo-code ahead)

//colors entered as 0-255 [R, G, B]colors = []; //holds final colors to be usedrand = new Random();//assumes n is less than 16,777,216randomGen(int n){   while (len(colors) &lt; n){      //ge

One way you could optimize this for better visibility would be to compare the distance between each new color and all the colors in the array:

for item in color{   itemSq = (item[0]^2 + item[1]^2 + item[2]^2])^(.5);   tempSq = (temp[0]^2 + temp[1]^2 + temp[2]^2])^(.5);   dist = itemSq - tempSq;   dist = abs(dist);}//NUMBER can be yo

But this approach would significantly slow down your algorithm.

Another way would be to scrap the randomness and systematically go through every 4 values and add a color to an array in the above example.

The text after removal of white spaces is: I've read somewhere the human eye can't distinguish between less than 4 values apart. so This is something to keep in mind. The following algorithm

I'm not sure this is exactly what you want, but this is one way to randomly generate non-repeating color values:

(beware, inconsistent pseudo-code ahead)

//colors entered as 0-255 [R, G, B]colors = []; //holds final colors to be usedrand = new Random();//assumes n is less than 16,777,216randomGen(int n){   while (len(colors) &lt; n){      //ge

One way you could optimize this for better visibility would be to compare the distance between each new color and all the colors in the array:

for item in color{   itemSq = (item[0]^2 + item[1]^2 + item[2]^2])^(.5);   tempSq = (temp[0]^2 + temp[1]^2 + temp[2]^2])^(.5);   dist = itemSq - tempSq;   dist = abs(dist);}//NUMBER can be yo

But this approach would significantly slow down your algorithm.

Another way would be to scrap the randomness and systematically go through every 4 values and add a color to an array in the above example.
```



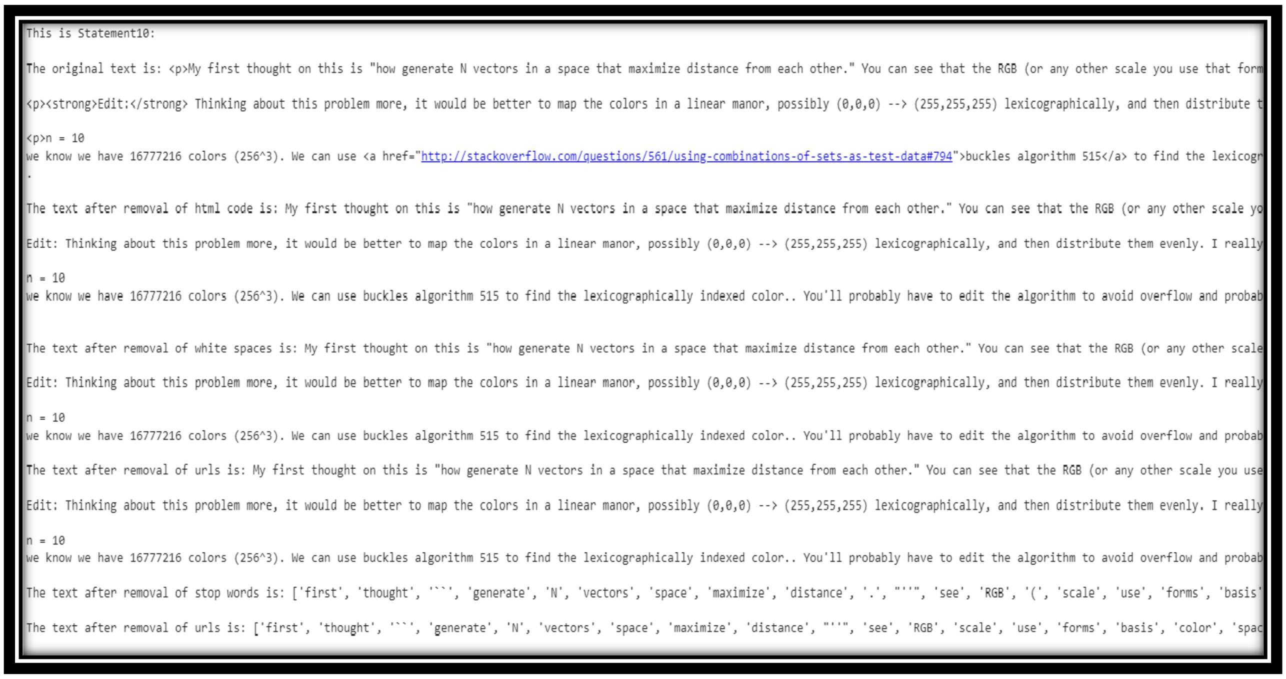
```
This is Statement6:

The original text is: <p>You might be able to use IronRuby for that. </p>

<p>Otherwise I'd suggest you have a directory where you place precompiled assemblies. Then you could have a reference in the DB to the assembly and class, and use reflection to load the proper

<p>If you really want to compile at run-time you could use the CodeDOM, then you could use reflection to load the dynamic assembly. <a href="http://msdn.microsoft.com/en-us/library/microsoft.

The text after removal of html code is: You might be able to use IronRuby for that.

Otherwise I'd suggest you have a directory where you place precompiled assemblies. Then you could have a reference in the DB to the assembly and class, and use reflection to load the proper as

If you really want to compile at run-time you could use the CodeDOM, then you could use reflection to load the dynamic assembly. MSDN article which might help.

The text after removal of white spaces is: You might be able to use IronRuby for that.

Otherwise I'd suggest you have a directory where you place precompiled assemblies. Then you could have a reference in the DB to the assembly and class, and use reflection to load the proper as

If you really want to compile at run-time you could use the CodeDOM, then you could use reflection to load the dynamic assembly. MSDN article which might help.

The text after removal of urls is: You might be able to use IronRuby for that.

Otherwise I'd suggest you have a directory where you place precompiled assemblies. Then you could have a reference in the DB to the assembly and class, and use reflection to load the proper as

If you really want to compile at run-time you could use the CodeDOM, then you could use reflection to load the dynamic assembly. MSDN article which might help.

The text after removal of stop words is: ['might', 'able', 'use', 'IronRuby', '.', 'Otherwise', "'d", 'suggest', 'directory', 'place', 'precompiled', 'assemblies', '.', 'could', 'reference',

The text after removal of urls is: ['might', 'able', 'use', 'IronRuby', 'Otherwise', "'d", 'suggest', 'directory', 'place', 'precompiled', 'assemblies', 'could', 'reference', 'DB', 'assembly',
```



```
This is Statement7:

The original text is: <p>You could use any of the DLR languages, which provide a way to really easily host your own scripting platform.</p>.

The text after removal of html code is: You could use any of the DLR languages, which provide a way to really easily host your own scripting platform.

The text after removal of white spaces is: You could use any of the DLR languages, which provide a way to really easily host your own scripting platform.

The text after removal of urls is: You could use any of the DLR languages, which provide a way to really easily host your own scripting platform.

The text after removal of stop words is: ['could', 'use', 'DLR', 'languages', ',', 'provide', 'way', 'really', 'easily', 'host', 'scripting', 'platform', '.']

The text after removal of urls is: ['could', 'use', 'DLR', 'languages', 'provide', 'way', 'really', 'easily', 'host', 'scripting', 'platform']



This is Statement8:

The original text is: <p>No, what you're doing is fine. Don't let those people confuse you.</p>

<p>If you've written the web services with .net then the reference proxies generated by .net are going to be quite suitable. The situation you describe (where you are both producer and consu

<p>If you need to connect to a web services that is <em>unknown</em> at compile time, then you would want a more dynamic approach, where you deduce the 'shape' of the web service. </p>

<p>But start by using the auto generated proxy class, and don't worry about it until you hit a limitation. And when you do -- come back to stack overflow ;-)</p>.

The text after removal of html code is: No, what you're doing is fine. Don't let those people confuse you.

If you've written the web services with .net then the reference proxies generated by .net are going to be quite suitable. The situation you describe (where you are both producer and consumer

If you need to connect to a web services that is unknown at compile time, then you would want a more dynamic approach, where you deduce the 'shape' of the web service.

But start by using the auto generated proxy class, and don't worry about it until you hit a limitation. And when you do -- come back to stack overflow ;-)

The text after removal of white spaces is: No, what you're doing is fine. Don't let those people confuse you.

If you've written the web services with .net then the reference proxies generated by .net are going to be quite suitable. The situation you describe (where you are both producer and consumer

If you need to connect to a web services that is unknown at compile time, then you would want a more dynamic approach, where you deduce the 'shape' of the web service.

But start by using the auto generated proxy class, and don't worry about it until you hit a limitation. And when you do -- come back to stack overflow ;-)
```



```
This is Statement9:

The original text is: <p>Isn't it also a factor which order you set up the colors?</p>

<p>Like if you use Dillie-Os idea you need to mix the colors as much as possible.
0 64 128 256 is from one to the next. but 0 256 64 128 in a wheel would be more "apart"</p>

<p>Does this make sense?</p>.

The text after removal of html code is: Isn't it also a factor which order you set up the colors?

Like if you use Dillie-Os idea you need to mix the colors as much as possible.
0 64 128 256 is from one to the next. but 0 256 64 128 in a wheel would be more "apart"

Does this make sense?

The text after removal of white spaces is: Isn't it also a factor which order you set up the colors?

Like if you use Dillie-Os idea you need to mix the colors as much as possible.
0 64 128 256 is from one to the next. but 0 256 64 128 in a wheel would be more "apart"

Does this make sense?

The text after removal of urls is: Isn't it also a factor which order you set up the colors?

Like if you use Dillie-Os idea you need to mix the colors as much as possible.
0 64 128 256 is from one to the next. but 0 256 64 128 in a wheel would be more "apart"

Does this make sense?

The text after removal of stop words is: ["n't", 'also', 'factor', 'order', 'set', 'colors', '?', 'Like', 'use', 'Dillie-Os', 'idea', 'need', 'mix', 'colors', 'much', 'possible', '.', '0', '6

The text after removal of urls is: ["n't", 'also', 'factor', 'order', 'set', 'colors', 'Like', 'use', 'Dillie-Os', 'idea', 'need', 'mix', 'colors', 'much', 'possible', '0', '64', '128', '256'
```



```
This is Statement10:

The original text is: <p>My first thought on this is "how generate N vectors in a space that maximize distance from each other." You can see that the RGB (or any other scale you use that form

<p><strong>Edit:</strong> Thinking about this problem more, it would be better to map the colors in a linear manor, possibly (0,0,0) --> (255,255,255) lexicographically, and then distribute t

<p>n = 10
we know we have 16777216 colors (256^3). We can use <a href="http://stackoverflow.com/questions/561/using-combinations-of-sets-as-test-data#794">buckles algorithm 515</a> to find the lexicogr
.

The text after removal of html code is: My first thought on this is "how generate N vectors in a space that maximize distance from each other." You can see that the RGB (or any other scale yo

Edit: Thinking about this problem more, it would be better to map the colors in a linear manor, possibly (0,0,0) --> (255,255,255) lexicographically, and then distribute them evenly. I really

n = 10
we know we have 16777216 colors (256^3). We can use buckles algorithm 515 to find the lexicographically indexed color.. You'll probably have to edit the algorithm to avoid overflow and probab


The text after removal of white spaces is: My first thought on this is "how generate N vectors in a space that maximize distance from each other." You can see that the RGB (or any other scale

Edit: Thinking about this problem more, it would be better to map the colors in a linear manor, possibly (0,0,0) --> (255,255,255) lexicographically, and then distribute them evenly. I really

n = 10
we know we have 16777216 colors (256^3). We can use buckles algorithm 515 to find the lexicographically indexed color.. You'll probably have to edit the algorithm to avoid overflow and probab

The text after removal of urls is: My first thought on this is "how generate N vectors in a space that maximize distance from each other." You can see that the RGB (or any other scale you use

Edit: Thinking about this problem more, it would be better to map the colors in a linear manor, possibly (0,0,0) --> (255,255,255) lexicographically, and then distribute them evenly. I really

n = 10
we know we have 16777216 colors (256^3). We can use buckles algorithm 515 to find the lexicographically indexed color.. You'll probably have to edit the algorithm to avoid overflow and probab

The text after removal of stop words is: ['first', 'thought', '``', 'generate', 'N', 'vectors', 'space', 'maximize', 'distance', '.', "''", 'see', 'RGB', '(', 'scale', 'use', 'forms', 'basis'

The text after removal of urls is: ['first', 'thought', '``', 'generate', 'N', 'vectors', 'space', 'maximize', 'distance', "''", 'see', 'RGB', 'scale', 'use', 'forms', 'basis', 'color', 'spac
```

Questions:

1. As discussed there might be need to add punctuations or retain URLs in the given Text.
   a. Write the sample python code for extracting text from audio and add appropriate punctuations to it
   b. Write a sample python code to identify the URLs from the text data and extract URLs from text data

A)

a)

Code:

```
import speech_recognition as sr
import string

recognizer = sr.Recognizer()
audio_file = "sample_audio.wav"
with sr.AudioFile(audio_file) as source:
   audio = recognizer.record(source)

try:
   text = recognizer.recognize_google(audio)
   print("Extracted Text:", text)
   text_with_punctuations = text + " " + string.punctuation
   print("Text with Punctuations:", text_with_punctuations)

except sr.UnknownValueError:
   print("Speech Recognition could not understand audio")
except sr.RequestError as e:
   print(f"Could not request results from Google Speech Recognition
service; {e}")
```

b)

Code:

```
import re
text = "Check out this website: https://travelog.surge.sh."

url_pattern = r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(\\),]|(?:%[0-9a-
fA-F][0-9a-fA-F]))+'

urls = re.findall(url_pattern, text)
for url in urls:
   print("Extracted URL:", url)
```

Outcomes:

CO2: Comprehend Words and Word Forms in NLP

Conclusion: (Conclusion to be based on the outcomes achieved)

We understood the concept of need of removal of punctuations, stop words, extra white spaces, URLs and HTML code from Text. We also implemented the same on a field of text from our dataset and observed the results.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. Allen.James, Natural Language Understanding, Benjamin Cumming, Second Edition, 1995
2. Jurafsky, Dan and Martin, James, Speech and Language Processing, Prentice Hall, 2008
3. Palash Goyal, Karan Jain, Sumit Pandey,Deep Learning for Natural Language Processing: Creating Neural Networks with Python, Apress, 2018