Text Data Cleaning

First steps in working with text data is to pre-process it. It is an essential step before the data is ready for analysis. Majority of available text data is **highly unstructured and noisy** in nature – to achieve better insights or to build better algorithms, it is necessary to play with clean data. For example, social media data is highly unstructured – it is an informal communication – typos, bad grammar, usage of slang, presence of unwanted content like URLs, Stop-words etc. we need to clean them.

Use Cases:

1. **Escaping Html Character:** Data obtained from web usually contains a lot of html entities like < > & which gets embedded in the original data.

```
st = "I luv my &lt;3 iphone &amp; you're awsm apple. DisplayIsAwesome, sooo happppppy 🙂 http://www.apple.com"
import html
html.unescape(s)
```

```
'I luv my <3 iphone & you're awsm apple. DisplayIsAwesome, sooo happppppy 🙂 http://www.apple.com'
```

2. **Removal of Stop-words**: When data analysis needs to be data driven at the word level, the commonly occurring words (stop-words) should be removed. One can either create a long list of stop-words or one can use predefined language specific libraries.

```
from nltk.corpus import stopwords
mess = """  You can check the target names (categories) and some data files by following commands. """
clean_mess = [word for word in mess.split() if word.lower() not in stopwords.words('english')]
```

```
clean_mess
```

```
['check',
 'target',
 'names',
 '(categories)',
 'data',
 'files',
 'following',
 'commands.']
```

3. **Removal of Punctuations**:

```
import string
mess = """  Twinkle, twinkle, little star, How I wonder what you are!    """
non_punc =[char for char in mess if char not in string.punctuation]
non_punc = "".join(non_punc)
non_punc
```

```
'  Twinkle twinkle little star How I wonder what you are    '
```

4. **Split Attached Words**: We humans in the social forums generate text data, which is completely informal in nature. Most of the tweets are accompanied with multiple attached words like RainyDay, PlayingInTheCold etc. These entities can be split into their normal forms using simple rules and regex.

```
import re
original_tweet = "I luv my &lt;3 iphone &amp; you're awsm apple. DisplayIsAwesome, sooo happppppy 🙂"
cleaned = " ".join(re.findall('[A-Z][^A-Z]*', original_tweet))
cleaned
```

```
'I luv my &lt;3 iphone &amp; you're awsm apple.  Display Is Awesome, sooo happppppy 🙂'
```

5. **Removel of URLs from text**

```
st2
```

```
'I luv my <3 iphone & you're awsm apple. DisplayIsAwesome, sooo happppppy 🙂 http://www.apple.com'
```

```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer('https?://(?:[-\w.]|(?:%[\da-fA-F]{2}))+', gaps = True)
tokenizer.tokenize(st2)
```

```
['I luv my <3 iphone & you're awsm apple. DisplayIsAwesome, sooo happppppy 🙂 ']
```

6. Stemming

A stemming algorithm reduces the words "working", "worked" to the root word, "work".

```
txt = "listen listening do doing work worked working"
from nltk.stem import PorterStemmer
st = PorterStemmer()
txt = " ".join([st.stem(w) for w in txt.split()])
txt
```

```
'listen listen do do work work work'
```