1.a

For the raw data, it may contain a lot of punctuation, shorthand, URL, etc.,these information are not important for us to understand the meaning of the text. It can be seen from the result of the tokenizer. Therefore, using regular expressions to replace these with null values before tokenizer is better.

```
def text_clean(data):
    text_no_special_entities = re.sub(r'\&\w*;|#\w*|@\w*', '', data)
    text_no_tickers = re.sub(r'\$\w*', '', text_no_special_entities)
    text_no_hyperlinks = re.sub(r'https?:\/\/.*\/\w*', '', text_no_tickers)
    text_no_small_words = re.sub(r'\b\w{1,2}\b', '', text_no_hyperlinks)
    text_no_whitespace = re.sub(r'\s\s+', ' ', text_no_small_words)
    text_no_whitespace = text_no_whitespace.lstrip(' ')
    text_no_punctuation = re.sub(r'\W', ' ', text_no_whitespace)
    return text_no_punctuation
```

data_token_**uncleaned**= ['RT', '@', 'Amila', '#', 'Test', 'Tom', "'s", 'newly', 'listed', 'Co', '&', 'amp', ';', 'Mary', "'s", 'unlisted', 'https', ':', '//t.co/xx34afsfQsh', 'In', 'this', 'week', "'", 's', 'lecture', 'you', 'learnt', 'about', 'modules', 'and', ...]

data_token_**cleaned**= ['Tom', 'newly', 'listed', 'Mary', 'unlisted', 'this', 'week', 'lecture', 'you', 'learnt', 'about', 'modules', 'and', 'packages', 'IPython, …]

1.b

The result after normalization and pos- tagger:

data_normalisation= ['tom', 'newly', 'listed', 'mary', 'unlisted', 'this', 'week', 'lecture', 'you', 'learnt', 'about', 'modules', 'and', 'packages', 'ipython', …]

postagged= [('tom', 'NN'), ('newly', 'RB'), ('listed', 'VBN'), ('mary', 'JJ'), ('unlisted', 'VBD'), ('this', 'DT'), ('week', 'NN'), ('lecture', 'NN'), ('you', 'PRP'), ('learnt', 'VBP'), ('about', 'IN'), ('modules', 'NNS'), ('and', 'CC'), ('packages', 'NNS'), ('ipython', 'VBP'), …]

1.c

Since the data has been normalized, all the letters are converted to lowercase, Since the data has been standardized, all the letters are converted to lowercase, resulting in incorrect tagging of some people's names. In addition, incorrect tagging in abbreviations for some words.

postagged= [('rt', 'NN'), **('@', 'NNP')**, **('amila', 'RB')**, ('#', '#'), ('test', 'NN'), ('tom',

'NN'), ("'s", 'POS'), ('newly', 'RB'), ('listed', 'VBN'), **('co', 'NN')**, ('&', 'CC'), **('amp', 'NN')**, (';', ':'), **('mary', 'CC')**, ("'s", 'POS'), ('unlisted', 'JJ'), **('https', 'NN')**, (':', ':'), **('//t.co/x34afsfqsh', 'NN')**, ('in', 'IN'), ('this', 'DT'), ('week', 'NN'), ("'", 'NNP'), …]

If the raw data are cleaned in the first step, the mistake of pos-tagging will decrease a lot. But also existing some incorrect tagging.

postagged= [('tom', 'NN'), ('newly', 'RB'), ('listed', 'VBN'), **('mary', 'JJ')**, ('unlisted', 'VBD'), ('this', 'DT'), ('week', 'NN'), ('lecture', 'NN'), ('you', 'PRP'), ('learnt', 'VBP'), ('about', 'IN'), ('modules', 'NNS'), ('and', 'CC'), ('packages', 'NNS'), **('ipython', 'VBP')**, ('and', 'CC'), …]

2.a

After the Porter Stemming, the result is:

word_stemmed= ['RT', '@', 'amila', '#', 'test', 'tom', "'s", '**newli**', 'list', 'Co', '&', 'amp', ';', '**mari**', "'s", 'unlist', 'http', ':', '//t.co/x34afsfqsh', 'In', 'thi', 'week', "'", 's', '**lectur**', 'you', 'learnt', 'about', '**modul**', 'and', '**packag**', ',', 'ipython', 'and', '**numpi**', '.', 'In', …]

It can be seen from the result, the terminal -y are turned to -i for some words, like newly, mary and numpy. And also, the terminal -e is been taken off.

2.b

The result of these steps(tokenizer,pos-tagging and lemmatize) is :

word_lemmatized= ['rt', '@', 'amila', '#', 'test', 'tom', "'s", 'newly', 'list', 'co', '&', 'amp', ';', 'mary', "'s", 'unlisted', 'http', ':', '//t.co/x34afsfqsh', 'in', 'this', 'week', "'", 's', 'lecture', 'you', 'learn', 'about', 'module', 'and', 'package', ',', 'ipython',...]

From the result, it can be seen that lemmatization can better handle irregularly transformed words. But, there is a problem about it, it can't recognise POS-based differences and the part-of -speech tagged from pos-tag can't be used directly. It must convert these part-of-speech to ADJ, ADV, NOUN and VERB through wordnet.

```
def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wn.ADJ
    elif treebank_tag.startswith('V'):
        return wn.VERB
    elif treebank_tag.startswith('N'):
        return wn.NOUN
    elif treebank_tag.startswith('R'):
```

```
        return wn.ADV
    else:
        return wn.NOUN
```

2.c

The result from Porter Stemming:

```
word_stemmed= ['rt', '@', 'amila', '#', 'test', 'tom', "'s", 'newli', 'list', 'co', '&', 'amp', ';', 'mari', "'s", 'unlist', 'http', ':', '//t.co/x34afsfqsh', 'in', 'thi', 'week', '”', 's', 'lectur', 'you', 'learnt', 'about', 'modul', 'and', 'packag', ',', 'ipython', 'and', 'numpi',...]
```

The result from lemmatization:

```
word_lemmatized= ['rt', '@', 'amila', '#', 'test', 'tom', "'s", 'newly', 'list', 'co', '&', 'amp', ';', 'mary', "'s", 'unlisted', 'http', ':', '//t.co/x34afsfqsh', 'in', 'this', 'week', '”', 's', 'lecture', 'you', 'learn', 'about', 'module', 'and', 'package', ',', 'ipython', 'and', 'numpy', '.', 'in', 'this', 'week', '”', 's', 'assess', 'exercise', 'you', 'will', 'need', 'to', 'use', 'numpy', …]
```

As can be seen from the results, lemmatization is better for extracting stems than porter stemming. This is because lemmatization deals better with irregular plurals and it can correctly deal with the terminal -y and -e for some words.

3.

For the remote webpage can use urllib to request and get a beautifulsoup class object, and then calling the properties of beautifulsoup class object to get the contents of title, summary and body.

```
raw_html = urllib.request.urlopen(url).read()
soup = bs4.BeautifulSoup(raw_html, features='lxml')
title = soup.title
summary = soup.summary
body = soup.body.get_text(strip=True)
```