Lab1-Assignment

Copyright: Vrije Universiteit Amsterdam, Faculty of Humanities, CLTL

This notebook describes the assignment for Lab 1 of the text mining course.

Points: each exercise is prefixed with the number of points you can obtain for the exercise.

We assume you have worked through the following notebooks:

- **Lab1.1-introduction**
- **Lab1.2-introduction-to-NLTK**
- **Lab1.3-introduction-to-spaCy**

In this assignment, you will process an English text (**Lab1-apple-samsung-example.txt**) with both NLTK and spaCy and discuss the similarities and differences.

Credits

The notebooks in this block have been originally created by Marten Postma. Adaptations were made by Filip Ilievski.

Tip: how to read a file from disk

Let's open the file **Lab1-apple-samsung-example.txt** from disk.

In [1]:
```
from pathlib import Path
```

In [2]:
```
cur_dir = Path().resolve() # this should provide you with the folder in which this notebook is placed
path_to_file = Path.joinpath(cur_dir, 'Lab1-apple-samsung-example.txt')
print(path_to_file)
print('does path exist? ->', Path.exists(path_to_file))
```

```
C:\Users\imps\github\ba-text-mining\lab_sessions\lab1\Lab1-apple-samsung-example.txt
does path exist? -> True
```

If the output from the code cell above states that **does path exist? -> False**, please check that the file **Lab1-apple-samsung-example.txt** is in the same directory as this notebook.

In [3]:
```
with open(path_to_file) as infile:
    text = infile.read()

print('number of characters', len(text))
```

```
number of characters 1142
```

[total points: 4] Exercise 1: NLTK

In this exercise, we use NLTK to apply **Part-of-speech (POS) tagging**, **Named Entity Recognition (NER)**, and **Constituency parsing**. The following code snippet already performs sentence splitting and tokenization.

In [4]:
```
import nltk
from nltk.tokenize import sent_tokenize
from nltk import word_tokenize
```

In [5]:
```
sentences_nltk = sent_tokenize(text)
```

In [6]:
```
tokens_per_sentence = []
for sentence_nltk in sentences_nltk:
    sent_tokens = word_tokenize(sentence_nltk)
    tokens_per_sentence.append(sent_tokens)
```

We will use lists to keep track of the output of the NLP tasks. We can hence inspect the output for each task using the index of the sentence.

In [7]:
```
sent_id = 1
print('SENTENCE', sentences_nltk[sent_id])
print('TOKENS', tokens_per_sentence[sent_id])
```

```
SENTENCE The six phones and tablets affected are the Galaxy S III, running the new Jelly Bean system, the Galaxy Tab 8.9 Wifi tablet, the Galaxy Tab 2 10.1, Galaxy Rugby Pro and Galaxy S III mini.
TOKENS ['The', 'six', 'phones', 'and', 'tablets', 'affected', 'are', 'the', 'Galaxy', 'S', 'III', ',', 'running', 'the', 'new', 'Jelly', 'Bean', 'system', ',', 'the', 'Galaxy', 'Tab', '8.9', 'Wifi', 'tablet', ',', 'the', 'Galaxy', 'Tab', '2', '10.1', ',', 'Galaxy', 'Rugby', 'Pro', 'and', 'Galaxy', 'S', 'III', 'mini', '.']
```

[point: 1] Exercise 1a: Part-of-speech (POS) tagging

Use `nltk.pos_tag` to perform part-of-speech tagging on each sentence.

Use `print` to **show** the output in the notebook (and hence also in the exported PDF!).

In [8]:
```
pos_tags_per_sentence = []
for tokens in tokens_per_sentence:
    tagged = nltk.pos_tag(tokens)
    pos_tags_per_sentence.append(tagged)
```

In [9]:
```
print(pos_tags_per_sentence)
```


```
[[('https', 'NN'), (':', ':'), ('//www.telegraph.co.uk/technology/apple/9702716/Apple-Samsung-lawsuit-six-more-products-under-scrutiny.html', 'JJ'), ('Documents', 'NNS'), ('filed', 'VBN'), ('to', 'TO'), ('the', 'DT'), ('San', 'NNP'), ('Jose', 'NNP'), ('federal', 'JJ'), ('court', 'NN'), ('in', 'IN'), ('California', 'NNP'), ('on', 'IN'), ('November', 'NNP'), ('23', 'CD'), ('list', 'NN'), ('six', 'CD'), ('Samsung', 'NNP'), ('products', 'NNS'), ('running', 'VBG'), ('the', 'DT'), ('``', '``'), ('Jelly', 'RB'), ('Bean', 'NNP'), ("''", "''"), ('and', 'CC'), ('``', '``'), ('Ice', 'NNP'), ('Cream', 'NNP'), ('Sandwich', 'NNP'), ("''", "''"), ('operating', 'VBG'), ('systems', 'NNS'), (',', ','), ('which', 'WDT'), ('Apple', 'NNP'), ('claims', 'VBZ'), ('infringe', 'VB'), ('its', 'PRP$'), ('patents', 'NNS'), ('.', '.')], ...]
```


[point: 1] Exercise 1b: Named Entity Recognition (NER)

Use `nltk.chunk.ne_chunk` to perform Named Entity Recognition (NER) on each sentence.

Use `print` to **show** the output in the notebook (and hence also in the exported PDF!).

In [10]:
```
ner_tags_per_sentence = []
for tokens in pos_tags_per_sentence:
    entities = nltk.chunk.ne_chunk(tagged)
    ner_tags_per_sentence.append(entities)
```

In [11]:
```
print(ner_tags_per_sentence)
```


```
[Tree('S', [('https', 'NN'), (':', ':'), ('//www.telegraph.co.uk/technology/apple/9702716/Apple-Samsung-lawsuit-six-more-products-under-scrutiny.html', 'JJ'), ('Documents', 'NNS'), ('filed', 'VBN'), ('to', 'TO'), ('the', 'DT'), Tree('ORGANIZATION', [('San', 'NNP')]), ('Jose', 'NNP'), ...])]
```


[points: 2] Exercise 1c: Constituency parsing

Use the `nltk.RegexpParser` to perform constituency parsing on each sentence.

Use `print` to **show** the output in the notebook (and hence also in the exported PDF!).

In [12]:
```
constituent_parser = nltk.RegexpParser('''
NP: {<DT>? <JJ>* <NN>*} # NP
P: {<IN>}           # Preposition
V: {<V.*>}          # Verb
PP: {<P> <NP>}      # PP -> P NP
VP: {<V> <NP|PP>*}  # VP -> V (NP|PP)*''')
```

In [13]:
```
constituency_output_per_sentence = []
for tagged in pos_tags_per_sentence:
    parsed = constituent_parser.parse(tagged)
    constituency_output_per_sentence.append(parsed)
```

In [14]:
```
print(constituency_output_per_sentence)
```


```
[Tree('S', [Tree('NP', [('https', 'NN')]), (':', ':'), Tree('NP', [('//www.telegraph.co.uk/technology/apple/9702716/Apple-Samsung-lawsuit-six-more-products-under-scrutiny.html', 'JJ'), ('Documents', 'NNS')]), Tree('VP', [Tree('V', [('filed', 'VBN')]), ...])])]
```


Augment the RegexpParser so that it also detects Named Entity Phrases (NEP), e.g., that it detects *Galaxy S III* and *Ice Cream Sandwich*

In [15]:
```
constituent_parser_v2 = nltk.RegexpParser('''
NP: {<DT>? <JJ>* <NN>*} # NP
P: {<IN>}           # Preposition
V: {<V.*>}          # Verb
PP: {<P> <NP>}      # PP -> P NP
VP: {<V> <NP|PP>*}  # VP -> V (NP|PP)*
NEP: {<NNP>{2,} | <NNP>+ <CD>+}''')
```

In [16]:
```
constituency_v2_output_per_sentence = []
for tagged in pos_tags_per_sentence:
    parsed = constituent_parser_v2.parse(tagged)
    constituency_v2_output_per_sentence.append(parsed)
```

In [17]:
```
print(constituency_v2_output_per_sentence)
```


```
[Tree('S', [Tree('NP', [('https', 'NN')]), (':', ':'), Tree('NP', [('//www.telegraph.co.uk/technology/apple/9702716/Apple-Samsung-lawsuit-six-more-products-under-scrutiny.html', 'JJ'), ('Documents', 'NNS')]), ...])]
```


[total points: 1] Exercise 2: spaCy

Use Spacy to process the same text as you analyzed with NLTK.

In [18]:
```
import spacy
nlp = spacy.load('en_core_web_sm')
```

In [19]:
```
doc = nlp(text)
```

small tip: You can use **sents = list(doc.sents)** to be able to use the index to access a sentence like **sents[2]** for the third sentence.

Sentence Splitting

In [37]:
```
spacy_tokens_per_sentence = []
for sentence in doc.sents:
    sent_tokens = []
    for token in sentence:
        spacy_token = token.text
        sent_tokens.append(spacy_token)
    spacy_tokens_per_sentence.append(sent_tokens)

print(spacy_tokens_per_sentence)
```


```
[['https://www.telegraph.co.uk/technology/apple/9702716/Apple-Samsung-lawsuit-six-more-products-under-scrutiny.html', '\n', 'Documents', 'filed', 'to', 'the', 'San', 'Jose', 'federal', 'court', 'in', 'California', ...]]
```


In [38]:
```
iterator = 0
iterator_spacy = 0
for sentence_nltk in tokens_per_sentence:
    if iterator == 0:
        iterator_spacy += 1
    if sentence_nltk == spacy_tokens_per_sentence[iterator_spacy]:
        print('NLTK and SpaCy produced the same output for sentences with index {} for NLTK and {} for SpaCy'.format(iterator, iterator_spacy))
    else:
        print('Sentences with index {} for NLTK and index {} for Spacy are different'.format(iterator, iterator_spacy))
        print('Differences are {}\n'.format(set(sentence_nltk).difference(set(spacy_tokens_per_sentence[iterator_spacy]))))
    iterator += 1
    iterator_spacy += 1
```


```
Sentences with index 0 for NLTK and index 1 for Spacy are different
...
Differences are {"''", '//www.telegraph.co.uk/technology/apple/9702716/Apple-Samsung-lawsuit-six-more-products-under-scrutiny.html', '``', 'https'}

NLTK and SpaCy produced the same output for sentences with index 1 for NLTK and 2 for SpaCy

Sentences with index 2 for NLTK and index 3 for Spacy are different
...
Differences are {"''", '``'}

Sentences with index 3 for NLTK and index 4 for Spacy are different
...
Differences are set()

NLTK and SpaCy produced the same output for sentences with index 4 for NLTK and 5 for SpaCy

NLTK and SpaCy produced the same output for sentences with index 5 for NLTK and 6 for SpaCy

Differences are set()
```


Part-of-speech (POS) tagging

In [39]:
```
spacy_pos_tags_per_sentence = []
for sentence in doc.sents:
    pos_tags = []
    for token in sentence:
        tagged = (token.text, token.tag_)
        pos_tags.append(tagged)
    spacy_pos_tags_per_sentence.append(pos_tags)

print(spacy_pos_tags_per_sentence)
```


```
[[('https://www.telegraph.co.uk/technology/apple/9702716/Apple-Samsung-lawsuit-six-more-products-under-scrutiny.html', 'NNP'), ('\n', '_SP'), ('Documents', 'NNS'), ('filed', 'VBN'), ('to', 'IN'), ('the', 'DT'), ('San', 'NNP'), ('Jose', 'NNP'), ...]]
```


In [42]:
```
iterator = 0
iterator_spacy = 0
for sentence_nltk in pos_tags_per_sentence:
    if iterator == 0:
        iterator_spacy += 1
    if sentence_nltk == spacy_pos_tags_per_sentence[iterator_spacy]:
        print('NLTK and SpaCy produced the same output for sentences with index {} for NLTK and {} for SpaCy'.format(iterator, iterator_spacy))
    else:
        print('Sentences with index {} for NLTK and index {} for Spacy are different'.format(iterator, iterator_spacy))
        print('Differences are {}\n'.format(set(sentence_nltk).difference(set(spacy_pos_tags_per_sentence[iterator_spacy]))))
    iterator += 1
    iterator_spacy += 1
```


```
Sentences with index 0 for NLTK and index 1 for Spacy are different
...
Differences are {('tablets', 'NNS')}

Sentences with index 2 for NLTK and index 3 for Spacy are different
...
Differences are {("''", "''"), ('``', '``')}

Sentences with index 3 for NLTK and index 4 for Spacy are different
...
Differences are {('iPhone', 'NN'), ('(', '('), (')', ')'), ('BJ0.66bn', 'NN'), ('iPad', 'NN')}

Sentences with index 4 for NLTK and index 5 for Spacy are different
...
Differences are {('mobile', 'NN')}

Sentences with index 5 for NLTK and index 6 for Spacy are different
...
Differences are {('iPad', 'NN')}
```


Named Entity Recognition

In [61]:
```
for sentence in doc.sents:
    for entity in sentence.ents:
        print(entity.text, entity.label_)
    print()
```

```
San Jose GPE
California GPE
November 23 DATE
six CARDINAL
Samsung ORG
Jelly Bean WORK_OF_ART
Apple ORG

six CARDINAL
the Galaxy S III GPE
Jelly Bean ORG
the Galaxy Tab 2 10.1 ORG

eWmacted ORG
Apple ORG

August DATE
Samsung ORG
US GPE
Apple ORG
1.05bn MONEY
iPad ORG
iPhone ORG

Samsung ORG

UK GPE
Samsung ORG
Apple ORG
South Korean NORP
iPad ORG
```

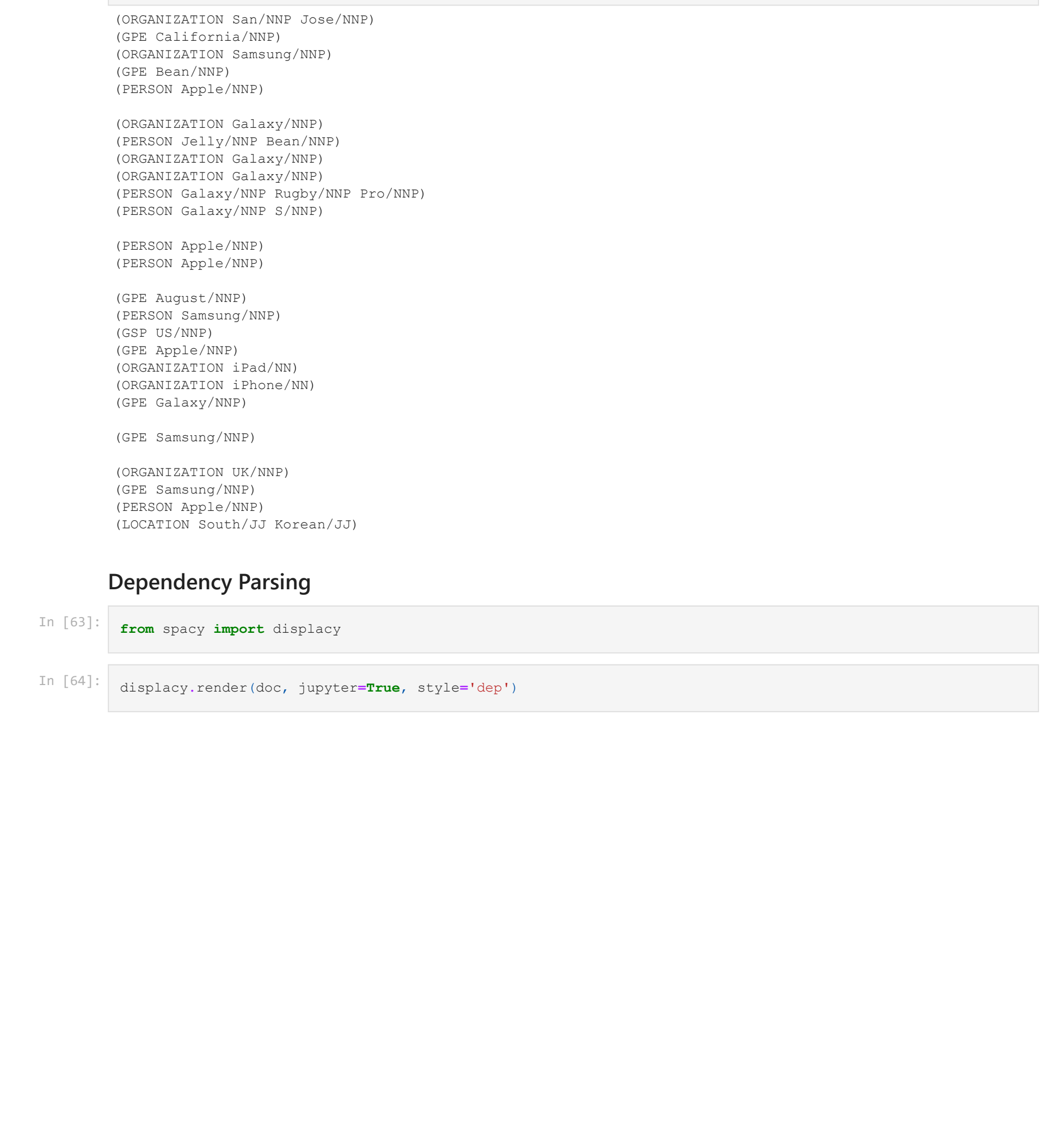
In [62]:
```
for sentence in ner_tags_per_sentence:
    for entity in sentence:
        if type(entity) != tuple:
            print(entity)
    print()
```

```
(ORGANIZATION San/NNP Jose/NNP)
(GPE California/NNP)
(ORGANIZATION Samsung/NNP)
(GPE Bean/NNP)
(PERSON Apple/NNP)

(ORGANIZATION Galaxy/NNP)
(PERSON Jelly/NNP Bean/NNP)
(ORGANIZATION Galaxy/NNP)
(ORGANIZATION Galaxy/NNP)
(PERSON Galaxy/NNP Rugby/NNP Pro/NNP)
(PERSON Galaxy/NNP S/NNP)

(PERSON Apple/NNP)
(PERSON Apple/NNP)

(GPE August/NNP)
(PERSON Samsung/NNP)
(GSP US/NNP)
(GPE Apple/NNP)
(ORGANIZATION iPad/NN)
(ORGANIZATION iPhone/NN)
(GPE Galaxy/NNP)

(GPE Samsung/NNP)

(ORGANIZATION UK/NNP)
(GPE Samsung/NNP)
(PERSON Apple/NNP)
(LOCATION South/JJ Korean/JJ)
```

Dependency Parsing

In [63]:
```
from spacy import displacy
```

In [64]:
```
displacy.render(doc, jupyter=True, style='dep')
```



[total points: 7] Exercise 3: Comparison NLTK and spaCy

We will now compare the output of NLTK and spaCy, i.e., in what do they differ?

[points: 3] Exercise 3a: Part of speech tagging

Compare the output from NLTK and spaCy regarding part of speech tagging.

- To compare, you probably would like to compare sentence per sentence. Describe if the sentence splitting is different for NLTK than for spaCy. If not, where do they differ?
- After checking the sentence splitting, select a sentence for which you expect interesting results and perhaps differences. Motivate your choice.
- Compare the output in `token.tag` from spaCy to the part of speech tagging from NLTK for each token in your selected sentence. Are there any differences? This is not a trick question; it is possible that there are no differences.

The sentence splitting in both NLTK and spaCy is very similar. The difference between the two can be found when splitting the link in the first sentence of the input, where spaCy tokenizes the entire link without breaking it down, on the other hand, NLTK breaks it down into 3 parts: "https", ':' and the rest of the link. Here spaCy seems to perform better since links should not be broken down in such way. Furthermore, as can be seen under the Sentence Splitting header, when using the .text attribute, SpaCy shows separator "\n" into a separate sentence. Also, spaCy and NLTK seem to have different characters for representation of the quotation marks ("``" -> for NLTK, and '""' -> for spaCy).

We took the first sentence. This is because, after checking the sentence splitting, it is the only sentence which has a major difference between the NLTK and spaCy with NLTK creating two additional splits than spaCy. This is caused by the URL that is present in this sentence.

There are quite a few differences between the two. Since NLTK splits the URL at the beginning into 3 parts, it tags 'https' with a NN tag and the rest of the link with a JJ tag, while spaCy just tags the entire URL with a NNP tag. Additionally, NLTK tags the word 'filed' as a 'VBN' while spaCy tags are as 'VBD' which is incorrect. Moreover, NLTK tags the first occurence of the word 'to' with the tag 'TO', while spaCy tags it as a preposition. Furthermore, while spaCy correctly tags the word 'Jelly' as a 'NNP', NLTK tags it as an adverb. Finally, spaCy correctly tags the word 'operating' in 'operating systems' as a 'NN' while NLTK incorrectly tags it as a verb.

[points: 2] Exercise 3b: Named Entity Recognition (NER)

- **Describe differences between the output from NLTK and spaCy for Named Entity Recognition. Which one do you think performs better?**

In NLTK, there are instances where Apple, Jelly Bean, Iphone, Galaxy are incorrectly recognized as persons. In spaCy however, they are recognized as organizations. Additionally, in NLTK the UK is recognized as an organization while spaCy recognizes it as a GPE. Furthermore, spaCy correctly recognizes that August is a date while NLTK incorrectly labels August as a GPE. Finally, spaCy correctly uses the NE NORP to label 'South Korean' where NLTK incorrectly labels it as a location and doesn't even utilize NORP anywhere. Furthermore, spaCy is able to recognize cardinal entities as well as measurements like money.

Generally, spaCy recognises more entities (in greater detail) than NLTK as it has more entity types, hence we think SpaCy performs better in that regard.

[points: 2] Exercise 3c: Constituency/dependency parsing

Choose one sentence from the text and run constituency parsing using NLTK and dependency parsing using spaCy.

- **describe briefly the difference between constituency parsing and dependency parsing**
- **describe differences between the output from NLTK and spaCy.**

Dependency parsing creates a graph that shows the simple (flat) dependencies/relationships between words in a sentence, while constituency parsing shows entire hierarchical relationships in phrases such as a NEP or a verb phrase.

The difference between the output from NLTK and spaCy is that the constituency parsing from NLTK detects hierarchical phrases within the sentence. For example, it detects 'is the world' as a verb phrase which also contains the noun phrase 'the world'. Another example is that the parser detects the noun phrase 'top mobile phone maker'. On the other hand, in the dependency parser from the spaCy, this type of hierarchy is not detected. However, unlike NLTK, in addition to the pos of each

word, relationships between individual words are visualized in a graph. For example, from the graph, we can see that the word 'maker' has the relation 'compound' with the word 'phone' (i.e., phone-maker). Another example we can see is that 'the' has a relation 'determiner' with the word 'world'.

End of this notebook