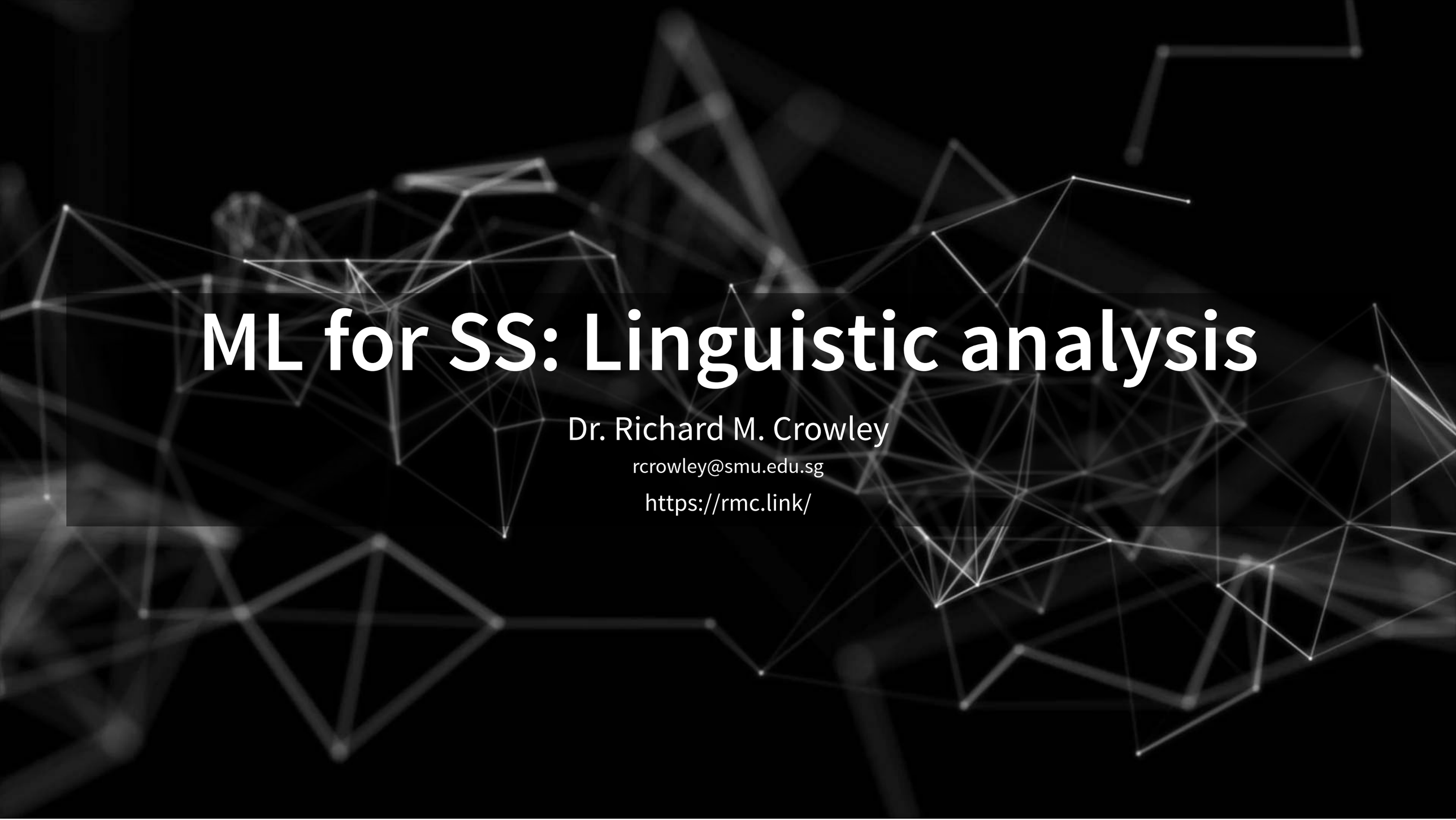


# ML for SS: Linguistic analysis

Dr. Richard M. Crowley

rcrowley@smu.edu.sg

https://rmc.link/

# Overview

# Papers

> Harris (1954)

- Outlines the main reason for why we do most of what we do in NLP

> Hope, Hu and Lu (2016)

- A use of Named Entity Recognition (NER) in a fairly simple way.
  - Names of persons, locations, or organizations
  - Percentages and monetary values; times and dates

> Garimella et al. (2019)

- Examines how social factors (gender) influence a common class of algorithms (tagging/parsing)

# Technical Discussion: Linguistics

## Python

- NLTK for standard/statistical approaches
- SpaCy for machine learning pipelines
- Stanza for Standford NLP methods
  - They have some interesting models for narrower uses
- BeautifulSoup for HTML parsing

## R

- Call python's SpaCy package from R using `spacyr`
- `rvest` for HTML parsing

> Python is generally a bit stronger for these topics, unless your data is clean and fairly small.

> Code using python, data, and dictionaries are on eLearn.

# Main application: Analyzing Wall Street Journal articles

- On eLearn you will find a full issue of the WSJ in text format

> Linguistic models using NLTK and SpaCy

- Tokenization and break documents into smaller chunks
- Part of speech tagging (grammar)
- Dependency parsing
- Named Entity Recognition (NER)
- Lemmatization


Using NLP parsers: NLTK

# NLTK

- NLTK stands for Natural Language Toolkit
- It provides a bunch of handy things for text analytics
  1. Corpora that are used in research and algorithm development
     - Tagged corpora are particularly valuable
  2. Models for things like dependency parsing
  3. Useful functions for working with text

# Setting up NLTK

- When using a resource from `NLTK`, we will often have install needed datasets

> Useful parts to download using `nltk.download()`

- `'punkt'`: Used for tokenizing words (splitting apart words in a document)
- `'brown'`: A corpus that contains part of speech information based on news articles
  - Can be used to train a part of speech tagger
- `'averaged_perceptron_tagger'`: An ML model for applying part of speech tags
- `'universal_tagset'`: If you only need simple part of speech labels, this is easier to work with
- `'treebank'`: Like `'brown'` above, but based on WSJ

# Tokenizing

```
text = 'Hedge funds are cutting their standard fees of 2% of assets under management and 20% of profits ami
tokens = nltk.tokenize.word_tokenize(text)
print(tokens)
```

```
['Hedge', 'funds', 'are', 'cutting', 'their', 'standard', 'fees', 'of', '2', '%', 'of', 'assets', 'under',
'management', 'and', '20', '%', 'of', 'profits', 'amid', 'pressure', 'from', 'investors', '.', 'A', 'team',
'of', 'Ares', 'and', 'CPP', 'Investment', 'Board', 'are', 'in', 'the', 'final', 'stages', 'of', 'talks', 'to',
'buy', 'luxury', 'retailer', 'Neiman', 'Marcus', 'for', 'around', '$', '6', 'billion', '.']
```

# Part of Speech tagging:

```
text = 'Hedge funds are cutting their standard fees of 2% of assets under management and 20% of profits ami
tokens = nltk.tokenize.word_tokenize(text)

# Requires previously running: nltk.download('brown')
brown_news_tagged = nltk.corpus.brown.tagged_sents(categories='news', tagset='brown')
pos_tagger = nltk.UnigramTagger(brown_news_tagged)

tagged1 = pos_tagger.tag(tokens)
print(tagged1)
```

```
[('Hedge', None), ('funds', 'NNS'), ('are', 'BER'), ('cutting', 'VBG'), ('their', 'PP$'), ('standard', 'JJ'),
('fees', 'NNS'), ('of', 'IN'), ('2', 'CD'), ('%', None), ('of', 'IN'), ('assets', 'NNS'), ('under', 'IN'),
('management', 'NN'), ('and', 'CC'), ('20', 'CD'), ('%', None), ('of', 'IN'), ('profits', 'NNS'), ('amid',
'IN'), ('pressure', 'NN'), ('from', 'IN'), ('investors', 'NNS'), ('.', '.'), ('A', 'AT'), ('team', 'NN'),
('of', 'IN'), ('Ares', None), ('and', 'CC'), ('CPP', None), ('Investment', 'NN-TL'), ('Board', 'NN-TL'),
('are', 'BER'), ('in', 'IN'), ('the', 'AT'), ('final', 'JJ'), ('stages', 'NNS'), ('of', 'IN'), ('talks',
'NNS'), ('to', 'TO'), ('buy', 'VB'), ('luxury', 'NN'), ('retailer', None), ('Neiman', None), ('Marcus', 'NP'),
('for', 'IN'), ('around', 'RB'), ('$', None), ('6', 'CD'), ('billion', 'CD'), ('.', '.')]
```

> Unigram tagging just uses the most common POS for a given token

# Part of Speech tagging: Neural network model

```
text = 'Hedge funds are cutting their standard fees of 2% of assets under management and 20% of profits ami
tokens = nltk.tokenize.word_tokenize(text)

# Requires previously running: nltk.download('averaged_perceptron_tagger')
tagged2 = nltk.pos_tag(tokens)
print(tagged2)
```

```
[('Hedge', 'NNP'), ('funds', 'NNS'), ('are', 'VBP'), ('cutting', 'VBG'), ('their', 'PRP$'), ('standard', 'JJ'),
('fees', 'NNS'), ('of', 'IN'), ('2', 'CD'), ('%', 'NN'), ('of', 'IN'), ('assets', 'NNS'), ('under', 'IN'),
('management', 'NN'), ('and', 'CC'), ('20', 'CD'), ('%', 'NN'), ('of', 'IN'), ('profits', 'NNS'), ('amid',
'IN'), ('pressure', 'NN'), ('from', 'IN'), ('investors', 'NNS'), ('.', '.'), ('A', 'DT'), ('team', 'NN'),
('of', 'IN'), ('Ares', 'NNS'), ('and', 'CC'), ('CPP', 'NNP'), ('Investment', 'NNP'), ('Board', 'NNP'), ('are',
'VBP'), ('in', 'IN'), ('the', 'DT'), ('final', 'JJ'), ('stages', 'NNS'), ('of', 'IN'), ('talks', 'NNS'), ('to',
'TO'), ('buy', 'VB'), ('luxury', 'NN'), ('retailer', 'NN'), ('Neiman', 'NNP'), ('Marcus', 'NNP'), ('for',
'IN'), ('around', 'IN'), ('$', '$'), ('6', 'CD'), ('billion', 'CD'), ('.', '.')]
```

> Unigram tagging just uses the most common POS for a given token

# Most common token/tag pairs

```
nltk.FreqDist(tagged2).most_common()
```

```
[(('of', 'IN'), 5), (('are', 'VBP'), 2), (('%', 'NN'), 2), (('and', 'CC'), 2), (('.', '.'), 2), (('Hedge',
'NNP'), 1), (('funds', 'NNS'), 1), (('cutting', 'VBG'), 1), (('their', 'PRP$'), 1), (('standard', 'JJ'), 1),
(('fees', 'NNS'), 1), (('2', 'CD'), 1), (('assets', 'NNS'), 1), (('under', 'IN'), 1), (('management', 'NN'),
1), (('20', 'CD'), 1), (('profits', 'NNS'), 1), (('amid', 'IN'), 1), (('pressure', 'NN'), 1), (('from', 'IN'),
1), (('investors', 'NNS'), 1), (('A', 'DT'), 1), (('team', 'NN'), 1), (('Ares', 'NNS'), 1), (('CPP', 'NNP'),
1), (('Investment', 'NNP'), 1), (('Board', 'NNP'), 1), (('in', 'IN'), 1), (('the', 'DT'), 1), (('final', 'JJ'),
1), (('stages', 'NNS'), 1), (('talks', 'NNS'), 1), (('to', 'TO'), 1), (('buy', 'VB'), 1), (('luxury', 'NN'),
1), (('retailer', 'NN'), 1), (('Neiman', 'NNP'), 1), (('Marcus', 'NNP'), 1), (('for', 'IN'), 1), (('around',
'IN'), 1), (('$', '$'), 1), (('6', 'CD'), 1), (('billion', 'CD'), 1)]
```

# What preceeds a noun?

```
text = 'Hedge funds are cutting their standard fees of 2% of assets under management and 20% of profits ami
tokens = nltk.tokenize.word_tokenize(text)

# Requires: nltk.download('brown') and nltk.download('universal_tagset')
brown_news_tagged = nltk.corpus.brown.tagged_sents(categories='news', tagset='universal')
pos_tagger = nltk.UnigramTagger(brown_news_tagged)
tagged3 = pos_tagger.tag(tokens)

bigrams = nltk.bigrams(tagged3)
noun_preceders = [a for (a, b) in bigrams if b[1] == 'NOUN']
fdist = nltk.FreqDist(noun_preceders)
fdist.most_common()
```

```
[(('of', 'ADP'), 3), (('Hedge', None), 1), (('standard', 'ADJ'), 1), (('under', 'ADP'), 1), (('amid', 'ADP'),
1), (('from', 'ADP'), 1), (('A', 'DET'), 1), (('CPP', None), 1), (('Investment', 'NOUN'), 1), (('final',
'ADJ'), 1), (('buy', 'VERB'), 1), (('Neiman', None), 1)]
```

# How are words and POS used

```
# Requires: nltk.download('treebank') and nltk.download('universal_tagset')
wsj_tagged = nltk.corpus.treebank.tagged_sents(tagset='universal')
pos_tagger = nltk.UnigramTagger(wsj_tagged)
tagged4 = pos_tagger.tag(tokens)
cfd_w2p = nltk.ConditionalFreqDist(tagged4)
cfd_p2w = nltk.ConditionalFreqDist([(item[1], item[0]) for item in tagged4])
```

```
cfd_w2p['funds']
```

```
FreqDist({'NOUN': 1})
```

```
cfd_p2w['NOUN']
```

```
FreqDist({'%': 2, 'funds': 1, 'fees': 1, 'assets': 1, 'management': 1, 'profits': 1, 'pressure': 1,
'investors': 1, 'team': 1, 'Investment': 1, ...})
```

# More details included in the Python file

- The python file uses a full issue of the WSJ
- Some code has to be adjusted to account for multiple files


Using NLP parsers: SpaCy

# SpaCy

- SpaCy provides a machine-learning based approach to many of the things NLTK does
- SpaCy is also perhaps a bit more user-friendly

```python
import spacy

# From outside of python, run: python -m spacy download en_core_web_sm
nlp = spacy.load("en_core_web_sm")
# pipes enabled by default: [tok2vec, tagger, parser, ner, attribute_ruler, lemmatizer]

text = """China said its exports rose 7.2% in August from a year earlier, the latest in a series of positiv

doc = nlp(text)
```

> `nlp()` runs all the pipes in one command

# Parse trees in SpaCy

- SpaCy has a visualization module called displaCy
- With this, we can quickly see how a sentence is structured
- To run it in a Jupyter notebook, use the below code:

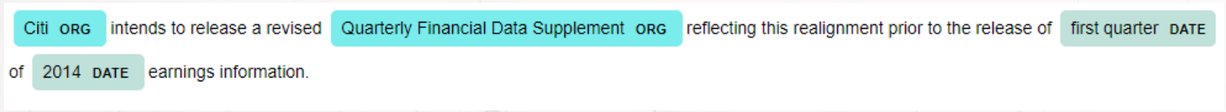
```
spacy.displacy.render(doc, style="dep", jupyter=True, options={'compact':True})
```

> Take a look at the code file to see the output

# NER: Named Entity Recognition

- During the `nlp()` call earlier, spaCy automatically did named entity recognition'
- Using an ML algorithm + the dependency tree, it tries to determine any proper nouns in the document
  - It also tries to label them
- You can visualize these as well with displayCy

```
spacy.displacy.render(sent, style="ent", jupyter=True)
```



# Extracting various components

## Entity extraction

```
doc.ents
```

```
(China, 7.2%, August, a year earlier)
```

```
for ent in doc.ents:
    print('"' + ent.text + '" is tagged as
```

```
"China" is tagged as "GPE" which comprises of
Countries, cities, states
"7.2%" is tagged as "PERCENT" which comprises
of Percentage, including "%"
"August" is tagged as "DATE" which comprises of
Absolute or relative dates or periods
"a year earlier" is tagged as "DATE" which
comprises of Absolute or relative dates or
periods
```

## Noun chunk extraction

```
list(doc.noun_chunks)
```

```
[China, its exports, August, a series, positive
economic reports]
```

## Lemmatization

```
doc[0].sent.lemma_
```

```
'China say its export rise 7.2% in August from
a year early, the late in a series of positive
economic report.'
```

# More details included in the Python file

- Using `nlp.pipe()` instead of `nlp()`
  - Allows you to apply a process to a corpus all at once (as a generator)
- Sentence boundary detection
- PoS tagging in SpaCy
- Lemmatization
- Extracting all entities from a corpus


Parsing HTML

# Overview

- As this part is code-heavy, we will do it in Jupyter
- The main idea is:
  1. Grab the main page of the website using `requests`
  2. Structure it with `beautifulsoup4` so we can traverse the page
  3. Grab the links to and names of standards, along with the publication years
  4. Traverse the links
  5. Extract the pdf locations from the traversed pages
  6. Grab the pdf files

# Addendum: Using R

- HTML files
  - You can load from a URL using `httr` or `RCurl`
  - You can use `XML` or `rvest` to parse out specific pieces of html files
- JSON files
  - You can process JSON data using `jsonlite`
- PDF files
  - Use `pdftools` to extract text into a vector of pages of text
  - Use `tabulizer` to extract tables straight from PDF files!
    - This is very painful to code by hand without this package
    - The package itself is a bit difficult to install, requiring Java and `rJava`, though

# Conclusion

# Wrap-up

> Linguistics is largely handled by importing specialized libraries

- NLTK for traditional measures
- SpaCy for more powerful, ML-based measures
- Stanza for Stanford NLP measures

> Easy to calculate many different measures, such as grammar/parts of speech or entities (NER)

# Packages used for these slides

**Python**

- `bs4`
- `nltk`
- `numpy`
- `requests`
- `spacy`

**R**

- downlit
- kableExtra
- knitr
- quarto
- reticulate
- revealjs

# References


- Garimella, Aparna, Carmen Banea, Dirk Hovy, and Rada Mihalcea. “Women’s syntactic resilience and men’s grammatical luck: Gender-bias in part-of-speech tagging and dependency parsing.” In Association for Computational Linguistics. 2019.
- Harris, Zellig S. “Distributional structure.” Word 10, no. 2-3 (1954): 146-162.
- Hope, Ole-Kristian, Danqi Hu, and Hai Lu. “The benefits of specific risk-factor disclosures.” Review of Accounting Studies 21, no. 4 (2016): 1005-1045.