Exercise 1, Python and spaCy. Install spaCy (`https://spacy.io/`) and use it to process this book: `https://www.gutenberg.org/files/57886/57886-0.txt`. Then, write code to calculate the following:

1. Number of sentences.
2. Number of tokens.
3. Average number of tokens per sentence.
4. Number of unique part-of-speech tags.
5. The counts of the most frequent part-of-speech tags (the top 5).
6. The counts of the least frequent part-of-speech tags (the bottom 5).
7. The number of named entities of each type (how many `person`s, `gpe`s, etc. are there?).
8. The number of sentences with at least one `neg` syntactic dependency.
9. The number of sentences whose root (in the dependency tree) is the verb *went*.

Push your code to the Git repository.

Exercise 2, Argument Identification. Given an input file with parse trees, implement code that does the following for each parse tree:

1. Loop through all the leaves and select the verbs. The part-of-speech tags of verbs start with the letter `V`.
2. For each verb, print the subtrees selected with the following heuristic:

```
1. current_node = verb
2. print siblings(current_node)
3. if current_node is not the root, current_node = parent(current_node)
4. go to 2
```

Use Python and the nltk (`https://www.nltk.org/`). In particular, you probably want to use `BracketParseCorpusReader` and `ParentedTree`. A complete implementation using `nltk` uses less than 100 lines of code. Push your code to the Git repository. Include a README file and the output of your implementation for the provided input file.

Sample input:

```
(S
  (S
    (NP-SBJ (DT The) (NN governor))
    (VP (MD could) (RB n't) (VP (VB make) (NP (PRP it)))))
  (, ,)
  (IN so)
  (S
    (NP-SBJ (DT the) (NN lieutenant) (NN governor))
    (VP (VBD welcomed) (NP (DT the) (JJ special) (NNS guests))))
  (. .))
```

and corresponding output:

```
*** Sentence 1:
(S
  (S
    (NP-SBJ (DT The) (NN governor))
    (VP (MD could) (RB n't) (VP (VB make) (NP (PRP it)))))
  (, ,)
  (IN so)
  (S
    (NP-SBJ (DT the) (NN lieutenant) (NN governor))
    (VP (VBD welcomed) (NP (DT the) (JJ special) (NNS guests))))
  (. .))
** Verb 1: (VB make)
* Subtree 1.1: it
* Subtree 1.2: n't
* Subtree 1.3: could
* Subtree 1.4: The governor
* Subtree 1.5: ,
* Subtree 1.6: so
* Subtree 1.7: the lieutenant governor welcomed the special guests
* Subtree 1.8: .
** Verb 2: (VBD welcomed)
* Subtree 2.1: the special guests
* Subtree 2.2: the lieutenant governor
* Subtree 2.3: so
* Subtree 2.4: ,
* Subtree 2.5: The governor could n't make it
* Subtree 2.6: .
```

Exercise 3, Building models. Your job is to build a model to predict whether somebody possesses something, and when with respect to a year.

1. Read this paper:

   Dhivya Chinnappa and Eduardo Blanco. 2018. Possessors Change Over Time: A Case Study with Artworks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Brussels, Belgium.
   Available at `http://www.cse.unt.edu/~blanco/papers/possessions_artworks.pdf`

2. The annotations and a corpus reader are given in `possessions_emnlp2018/`. Follow these steps:

   (a) Install Conda and restore the environment `conda.env`. Learn about Conda and Anaconda here: `https://conda.io/projects/conda/en/latest/index.html#`. If you install the packages listed in `conda.env`, the code provided to you will work.

   (b) Understand the code provided and the output of the reader:

   ```
   $ python corpus_reader.py emnlp_2018_annotations.csv
   ```

   The command above will take a while, you may want to only read a few instances to begin with:

   ```
   $ python corpus_reader.py emnlp_2018_annotations.csv -m 300
   ```

   (c) Write Python code to calculate the following:

   i. Number of Wikipedia articles
   ii. Average number of Wikipedia sections per article
   iii. Number of possessors and years per article (2 numbers)
   iv. Number of unique possessors and unique years per article (2 numbers)
   v. Number of unique (possessor, year) pairs (1 number)
   vi. All the numbers needed to draw Figures 2, 4, 5 and 6 from the paper. Write code to get the numbers yourself, do not just read the figures in the paper. Also, draw the figures yourself.

   (d) Build a model to solve the task. For each possessor and year, your model (or models) need to predict the labels for *before*, *during* and *after*. Use the *training* instances for training, *development* instances for validation, and *test* to get the actual results. You may use a few instances to get started, but when you have the code ready you should train with all instances to get better results. You cannot compare your results agains the ones reported in the paper unless you test with exactly the test instances.

   I recommend your first build a baseline with `scikit-learn`, and then you use `Keras` or `Tensorflow` to build a stronger model. Replicating the same neural network described in the paper is fine.

I recommend your start with `sample.py` to write your code. The examples here (all in Keras, `https://github.com/keras-team/keras/tree/master/examples`) will be very useful, especially `imdb_lstm.py`, `imdb_bidirectional_lstm.py`, `pretrained_word_embeddings.py`.