```
from IPython.display import HTML, display

def set_css():
  display(HTML('''
    <style>
        pre {
            white-space: pre-wrap;
        }
    </style>
  '''))
get_ipython().events.register('pre_run_cell', set_css)
```

Importing Libraries

```
!python -m spacy download en_core_web_sm
!pip install pdfminer.six
```

```
import numpy as np
import pandas as pd
from gensim.models import Word2Vec
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
import spacy
from spacy import displacy
import re
import string
from pdfminer.high_level import extract_text
import en_core_web_sm
from IPython.display import clear_output

nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nlp = en_core_web_sm.load()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Text extraction (PDF)

```
def pdf2text(n_files):
  pdfs = []
  for i in range(1, n_files + 1):
    pdfs.append(extract_text('job{}.pdf'.format(i), maxpages = 2))
  return pdfs
```

```
jobs = pdf2text(13)
```

Data cleaning and processing

Regex

```
alphabets = '([A-Za-z])'
prefixes = '(Mr|Mrs|Ms|Dr|Pr)[.]'
suffixes = '(Inc|Ltd|Jr|Sr|Co)'
acronyms = '(?:[a-zA-Z]\.){2,}'
websites = '[.](com|net|org|io|gov|fr|uk|usa|esp)'
starters = '(Mr\.|Mrs\.|Ms\.|Dr\.|Pr\.|However|But|The\s|This\s|That\s|Those\s|Their\s|
```

Lemmatization

```
lemmatizer = WordNetLemmatizer()

def nltk2wn_tag(nltk_tag):
  if nltk_tag.startswith('J'):
    return wordnet.ADJ
  elif nltk_tag.startswith('V'):
    return wordnet.VERB
  elif nltk_tag.startswith('N'):
    return wordnet.NOUN
  elif nltk_tag.startswith('R'):
    return wordnet.ADV
  else:
    return None

def lemmatize_sentence(sentence):
  nltk_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
  wn_tagged = map(lambda x: (x[0], nltk2wn_tag(x[1])), nltk_tagged)
  res_words = []
```

```
  res_words = []
  for word, tag in wn_tagged:
    if tag is None:
      res_words.append(word)
    else:
      res_words.append(lemmatizer.lemmatize(word, tag))
  return ' '.join(res_words)
```

Split function

```
def split_into_sentences(text):
  text = text.replace('\n', ' ')
  text = ' '.join([word for word in text.split(' ') if word != ''])
  text = text.replace('\uf0b7', '').replace('\x0c', '').replace('\uf054', '').replace('
  text = re.sub('\([a-zA-Z]{1}\)', '', text)
  text = re.sub(prefixes, '\\1<prd>', text)
  text = re.sub(suffixes + '[.]', '\\1<prd>', text)
  text = re.sub('[.]' + alphabets, '<prd>\\1', text)
  text = re.sub('([1-9a-zA-Z])[.]([1-9a-zA-Z])', '\\1<prd>\\2', text)
  text = re.sub(alphabets + '[.]' + alphabets, '\\1<prd>\\2', text)
  text = re.sub(alphabets + '[.]' + alphabets + '[.]' + alphabets, '\\1<prd>\\2<prd>\\3
  text = re.sub('\.{2,}', '<prd>', text)
  text = re.sub(websites, '<prd>\\1', text)
  text = re.sub(starters, '<stop>\\1', text)
  for acr in re.findall(acronyms, text): text = text.replace(acr, '<prd>'.join(acr.spli
  text = text.replace('.', '.<stop>').replace('!', '!<stop>').replace('?', '?<stop>').r
  text = text.replace('<prd>', '.')

  sentences = text.split('<stop>')
  sentences = [sent.strip() for sent in sentences]
  sentences = [sent for sent in sentences if sent not in ['', ' ']]
  sentences = [sent for sent in sentences if len(sent) > 1]
  sentences = list(map(lambda sent: ' '.join([word for word in sent.split() if word.low

  sentences = [lemmatize_sentence(sent) for sent in sentences]
  sentences = [sent.lower() for sent in sentences if sent != '']

  return sentences
```

Extracting sentences to a dataframe

```
sentences = []
docs = []
for i in range(13):
    sent_doc = split_into_sentences(jobs[i])
    sentences += sent_doc
    for k in range(len(sent_doc)): docs.append('Job {}'.format(i + 1))
```

```
df = pd.DataFrame()
df['Document'] = docs
df['Sentence'] = sentences
df.to_csv('sentences.csv')

df.head()
```

	Document	Sentence
0	Job 1	job vacancy notice
1	Job 1	software development engineer – time metrology
2	Job 1	international bureau weights measures (bipm)...
3	Job 1	bipm base sèvres , outskirts paris (france) ...
4	Job 1	information find www.bipm.org .

Cleaning sentences

```
def clean_sentence(sentence):
    sentence = re.sub('\.([a-z0-9])', '<dot>\\1', sentence)
    sentence = ' '.join([word for word in sentence.split() if word not in string.punctu
    sentence = sentence.replace('<dot>', '.')
    if ' @ ' in sentence:
        sentence = sentence.replace(' @ ', '@')
    if ' ’ ' in sentence:
        sentence = sentence.replace(' ’ ', '\'')

    return sentence

df['Sentence'] = df['Sentence'].apply(clean_sentence)
df.head()
```

	Document	Sentence
0	Job 1	job vacancy notice
1	Job 1	software development engineer – time metrology
2	Job 1	international bureau weights measures bipm int...
3	Job 1	bipm base sèvres outskirts paris france intern...
4	Job 1	information find www.bipm.org

Tagging important chunks

```
skills = list(pd.read_table('skills.txt', sep = ',').columns)
job_titles = list(pd.read_csv('job_titles.txt').columns)
job_titles = [title.lower() for title in job_titles]
```

```
contract_types = ['full-time', 'part-time', 'fixed-term', 'temporary', 'internship']
degrees = ['associate degree', "bachelor's degree", "master's degree", 'doctoral degree

def tag_words(sentence):
    URL = '[^@](((https?):\/\/)?(www.)[a-z0-9]+\.[a-z]{2,})'
    NUMBER = '\s([1-9]{1,})'
    EMAIL = '([a-zA-Z0-9_\.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-\.]+)'

    #EMAIL
    sentence = re.sub(EMAIL, '<email>\\1</email>', sentence)

    # URL
    sentence = re.sub(URL, ' <url>\\1</url>', sentence)

    # JOB TITLES
    sorted_list = sorted(job_titles, key = len)
    sorted_list.reverse()
    for title in sorted_list:
        if title in sentence:
            sentence = sentence.replace(title, ' <job_title>{}</job_title>'.format(titl
            break

    # SKILLS
    sentence = sentence.split()
    sentence = list(map(lambda word: ' <skill>{}</skill>'.format(word) if word in skill
    sentence = ' '.join(sentence)

    # COUNTRIES, CITIES, STATES
    doc = nlp(sentence)
    for ent in doc.ents:
        if ent.label_ == 'GPE':
            sentence = sentence.replace(ent.text, '<loc>{}</loc>'.format(ent.text))

    # DATE
    doc = nlp(sentence)
    for ent in doc.ents:
        if (ent.label_ == 'DATE') & ('<' not in ent.text) & ('>' not in ent.text):
            sentence = sentence.replace(ent.text, ' <date>{}</date>'.format(ent.text))

    # DEGREE
    for degree in degrees:
        if degree in sentence:
            sentence = sentence.replace(degree, ' <degree>{}</degree>'.format(degree))
            break

    # COMPANY
    doc = nlp(sentence)
    for ent in doc.ents:
        if ent.label_ == 'ORG':
            sentence = sentence.replace(ent.text, '<company>{}</company>'.format(ent.te

    # CONTRACT_TYPE
    sentence = sentence.split()
    sentence = list(map(lambda word: ' <contract_type>{}</contract_type>'.format(word)
    sentence = ' '.join(sentence)
```

```
59
60     return sentence
```

```
df['Sentence'] = df['Sentence'].apply(tag_words)
df.head()
```

	Document	Sentence
0	Job 1	job vacancy notice
1	Job 1	<skill>software</skill> development engineer –...
2	Job 1	<company>international bureau weights measures...
3	Job 1	bipm base sèvres outskirts <company>paris fran...
4	Job 1	information find <url>www.bipm.org</url>

Extracting tags

```
def extract_tag(tag, sentence):
    elements = []
    for element in sentence.split('{}>'.format(tag)):
        if element.endswith('</'):
            elements.append(element.replace('</', ''))
    elements = [element for element in elements if element != '']
    if len(elements) > 0: return set(elements)
```

```
df['Email'] = [extract_tag('email', sent) for sent in df['Sentence'].values]
df['URL'] = [extract_tag('url', sent) for sent in df['Sentence'].values]
df['Job Title'] = [extract_tag('job_title', sent) for sent in df['Sentence'].values]
df['Skills'] = [extract_tag('skill', sent) for sent in df['Sentence'].values]
df['Location'] = [extract_tag('loc', sent) for sent in df['Sentence'].values]
df['Company'] = [extract_tag('company', sent) for sent in df['Sentence'].values]
df['Date'] = [extract_tag('date', sent) for sent in df['Sentence'].values]
df['Degree'] = [extract_tag('degree', sent) for sent in df['Sentence'].values]
df['Contract Type'] = [extract_tag('contract_type', sent) for sent in df['Sentence'].va

df.head()
```

	Document	Sentence	Email	URL	Job Title	Sk
0	Job 1	job vacancy notice	None	None	None	

Creating model dataframe

measures...

```
df_model = df.copy()
df_model.drop(columns = ['Email', 'URL', 'Date', 'Degree', 'Contract Type'], inplace =
df_model['Job Title'] = df_model['Job Title'].apply(lambda x: 0 if x is None else 1)
df_model['Skills'] = df_model['Skills'].apply(lambda x: 0 if x is None else 1)
df_model['Location'] = df_model['Location'].apply(lambda x: 0 if x is None else 1)
df_model['Company'] = df_model['Company'].apply(lambda x: 0 if x is None else 1)
df_model['Sentence'] = df_model['Sentence'].str.replace('<.*>', '', regex = True)

df_model.head()
```

	Document	Sentence	Job Title	Skills	Location
0	Job 1	job vacancy notice	0	0	0
1	Job 1	development engineer – time metrology	0	1	0
2	Job 1	whose mandate provide basis coherent system m...	0	0	0
3	Job 1	bipm base sèvres outskirts staff 70	0	0	0
4	Job 1	information find	0	0	0

```
1
```