DATA X

NLP Module: Text Processing

Berkeley SCET

What is NLP?

Natural Language Processing

- Analyzes language and extracts meaning

Multiple Uses :

- Sentiment analysis
- Text Classification
- Natural language generation
- Automatic Captioning
- Machine Translation
- And More!

NLP Process

Text Processing

Clean up the text to make it easier to use and more consistent to increase prediction accuracy later on

Feature Engineering & Text Representation

Learn how to extract information from text

Learning Models

Use learning models to identify parts of speech, entities, sentiment, and other aspects of the text.

Cleaning Text Using Built in Str Methods

Importance of Cleaning

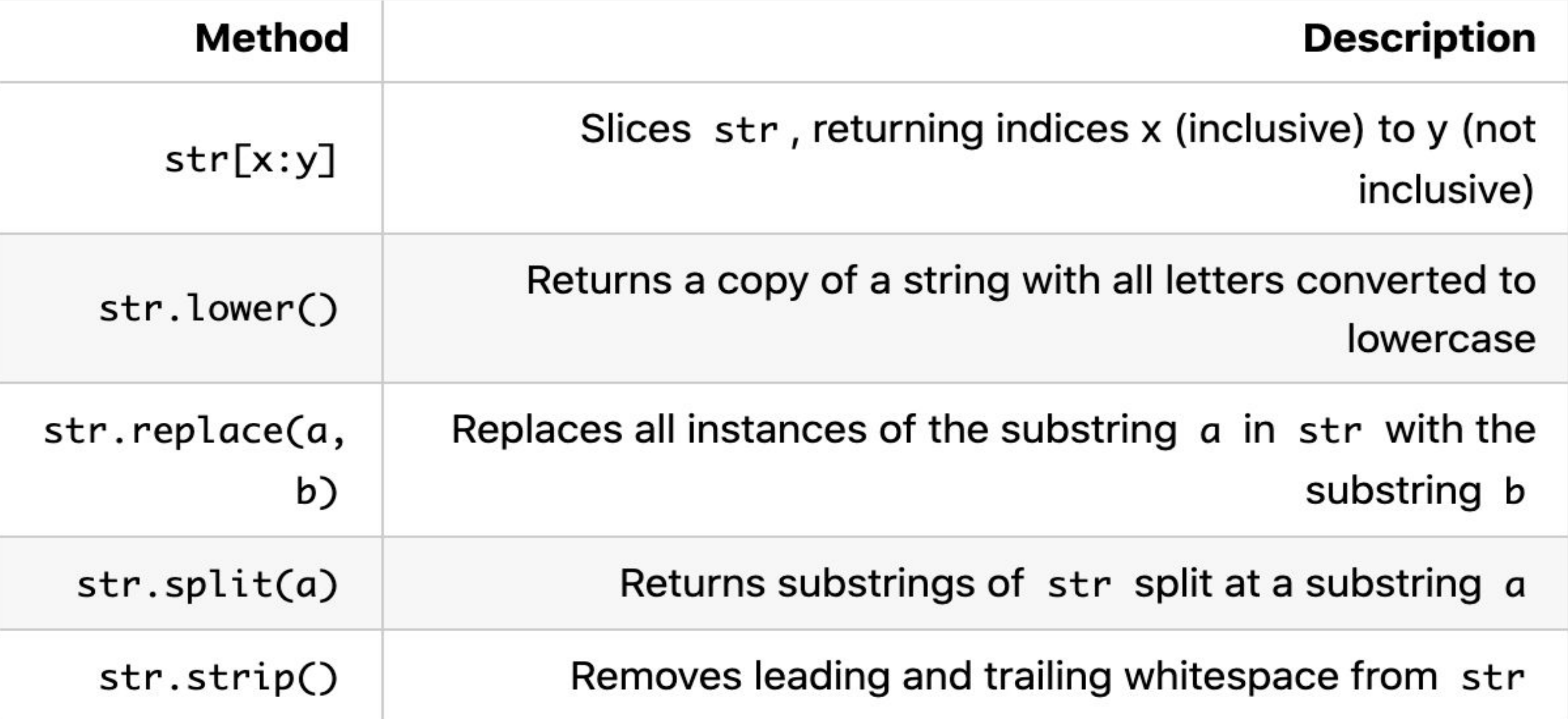
Datapoints have different syntax, need to have the same format to increase accuracy of nlp

Need to look through data first to see what to clean

Some Differences to Check For:

- Capitalization: `qui` vs `Qui`
- Different punctuation conventions: `St.` vs `St`
- Omission of words: `County/Parish` is absent in the `population` table
- Use of whitespace: `DeWitt` vs `De Witt`
- Different abbreviation conventions: `&` vs `and`

Methods Useful for Cleaning



Method	Description
`str[x:y]`	Slices `str`, returning indices x (inclusive) to y (not inclusive)
`str.lower()`	Returns a copy of a string with all letters converted to lowercase
`str.replace(a, b)`	Replaces all instances of the substring `a` in `str` with the substring `b`
`str.split(a)`	Returns substrings of `str` split at a substring `a`
`str.strip()`	Removes leading and trailing whitespace from `str`

Cleaning Text Using Regular Expressions (Regex)

Intro to Regex

Allows us to create general patterns for strings

Literals:

- A literal character in a regular expression matches the character itself. For example, the regex `r"a"` will match any `"a"` in the string.

Characters with Special Meaning:

- **Period character '.'** : matches any character that contains the character after the period
  - `show_regex_match("Call me at 382-384-3840.", r".all")`
  - `**Call** me at 382-384-3840.`
- **Backslash character '\'**: signals to interpret the next character literally
  - `show_regex_match("Call me at 382-384-3840.", r"\.")`
  - `Call me at 382-384-3840.`
- **Period character '.'**: match parts of pattern that may vary
  - `show_regex_match("Call me at 382-384-3840.", "...-...-....")`
  - `Call me at **382-384-3840**.`

Intro to Regex Cont.

Negating Characters:

- A negated character class matches any character except the characters in the class. To create a negated character class, wrap the negated characters in `[^ ]`

Square Brackets:

- **[x]:** Square brackets match something that you kind of don’t know about a string you’re looking for
  - `[DB]an - matches ‘Dan’ & ‘Ban’`
- **[x-x]**: You specify a range by writing the first character, followed by a dash, and ending with the last character
  - `[A-Z]an - matches ‘Aan’, ‘Ban’, ‘Can’, ‘Dan’, ... ‘Zan’`
  - `[0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9] - matches 3 digits, a dash, 3 more digits, a dash, and 4 more digits (phone number)`

Regex Methods Useful for Text Processing

re.search

`re.search(pattern, string)` searches for a match of the regex `pattern` anywhere in `string`. It returns a truthy match object if the pattern is found; it returns `None` if not.

re.findalll

`re.findall(pattern, string)` extracts substrings that match a regex. This method returns a list of all matches of `pattern` in `string`.

re.sub

`re.sub(pattern, replacement, string)` replaces all occurrences of `pattern` with `replacement` in the provided `string`. This method behaves like the Python string method `str.sub` but uses a regex to match patterns.

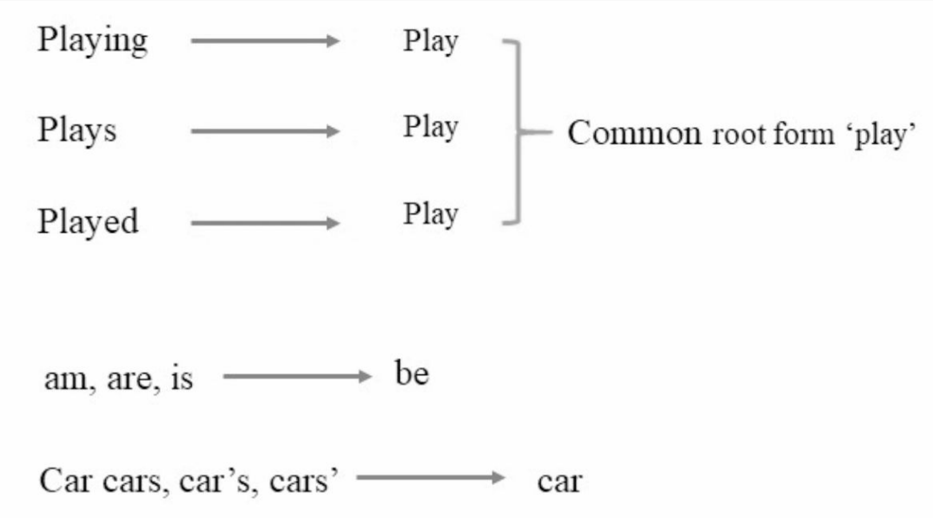
re.split

`re.split(pattern, string)` splits the input `string` each time the regex `pattern` appears. This method behaves like the Python string method `str.split` but uses a regex to make the split.

Stemming & Lemmatization

What is Stemming?

"Stemming is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the Language."



Playing → Play

Plays → Play

Played → Play

Common root form ‘play’

am, are, is → be

Car cars, car’s, cars’ → car

Stemming in Python

Nltk.stem has different types of stemmers that all vary slightly in how they stem and the rules that they follow:

1. Import a stemmer “from nltk.stem import PorterStemmer”

2. Iterate through data and iterate through each word in the datapoint and take each word and stem it using porter.stem(word) and then rejoin words **this is because the stemmer works only on a per word bases and will just return the original sentence if you pass sentence into porter.stem()

What is Lemmatization?

Lemmatization, unlike Stemming, reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called ***Lemma****. A lemma (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words.*

-For example, *runs, running, ran* are all forms of the word *run*, therefore *run* is the lemma of all these words. Because lemmatization returns an actual word of the language, it is used where it is necessary to get valid words.

Lemmatizing in Python

1. Import a lemmatizer “from nltk.stem import WordNetLemmatizer”

2. Iterate through data and iterate through each word in the datapoint and take each word and stem it using wordnet_lemmatizer.lemmatize(“word”) and then rejoin words **this is bc the stemmer works only on a per word bases and will just return the original sentence if you pass sentence into porter.stem()

Tokenization & Removing Stopwords

Importance of Removing Stopwords:

- Stopwords are words like “a” “the” “you” which don’t add much external meaning to sentences especially when classifying

What is Tokenization?:

- Breaking up words in a sentence to individual words

Removing Stopwords from a Sentence

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
example_sent = "This is a sample sentence, showing off the stop words filtration."
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(example_sent)
filtered_sentence = [w for w in word_tokens if not w in stop_words]
 filtered_sentence = []
  for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)
```