Natural languages: are the normal languages we use to communicate in everyday life, e.g. English, Chinese... etc., which -unlike languages like Python- have been naturally developed through time.

So, when we talk about Natural Language Processing (NLP), we mean how computers process these languages; or simply put, how to deal with text data.

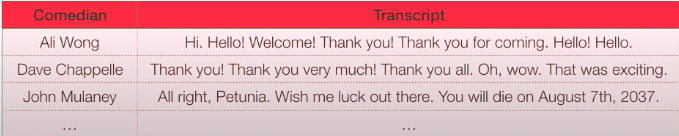
**Some of the most important examples of NLP are:**

* **Sentiment Analysis**: deciding whether a statement carries a positive or a negative emotion.
* **Topic Modeling:** classifying different texts based on their topic.
* **Text Generation:** creating new texts that are similar to previously written ones.

To solve a problem using NLP, we first need to start by gathering raw data about the subject and putting it into one of the standard formats that are easily analyzed and processed by computers. **Different types of analysis require different data formats, some of which are:**

* **Corpus**

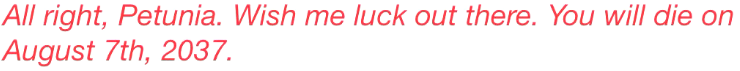
A corpus is a collection of texts, in which we put our data in an organized table. We usually create this corpus using Pandas library with the DataFrame function.



* **Document-term Matrix**

For us to put the data in this form, we need to clean it, tokenize it, then put it into a matrix.

For example:



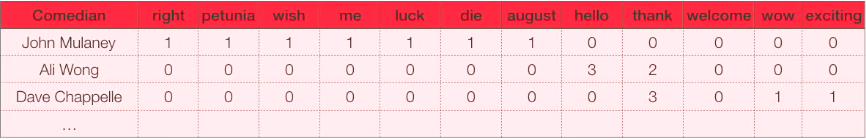
* **Cleaning**: by removing punctuation, numbers and write all letters in lowercase.

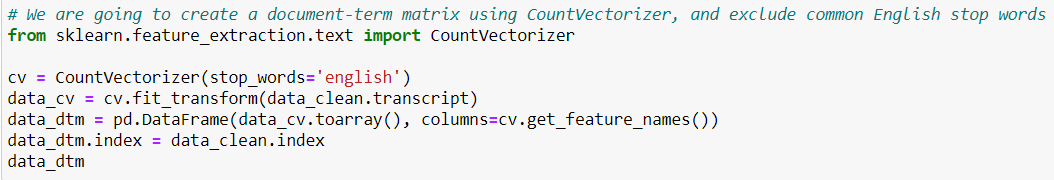
(You can do this by using Regular expression (re) operations in Python)

* **Tokenization**: is to split text into smaller pieces. The most common token size is a word. It can also be a sentence.

Now that every word is its own token, you can filter out words that have very little meaning (the, out, all…etc.) – these are called **stop words.**

After that, we end up with the essential words out of each sentence, each represented as its own token. This representation of text is called a **bag of words model.** It is a simple format that ignores order.

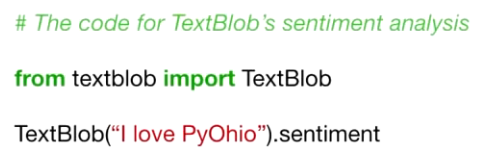
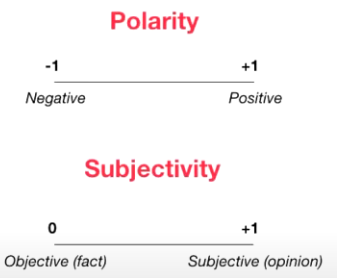
* **Matrix**: we put the final words into a matrix. And the reason we need to do that is because we need to store this info for multiple documents.

 We can easily make this matrix using CounterVectorizer() function in the scikit- learn library.

Back to the NLP techniques:

* **Sentiment Analysis**

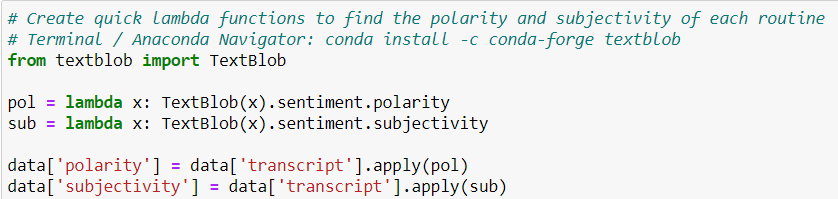
For Sentiment Analysis, we should input our data as a corpus. The reason we do not use document-term matrix (bag of words format) here is because order matters, i.e. “great” = positive. “not great” = negative.

We then use the python library ***TextBlob*** (built on nltk) to give us a sentiment score (how positive/negative) and a subjectivity score (how opinionated) for a sentence. It does that by finding all the words and phrases that it can assign a polarity and subjectivity to, and averages all of them together.

#Output:



Example:

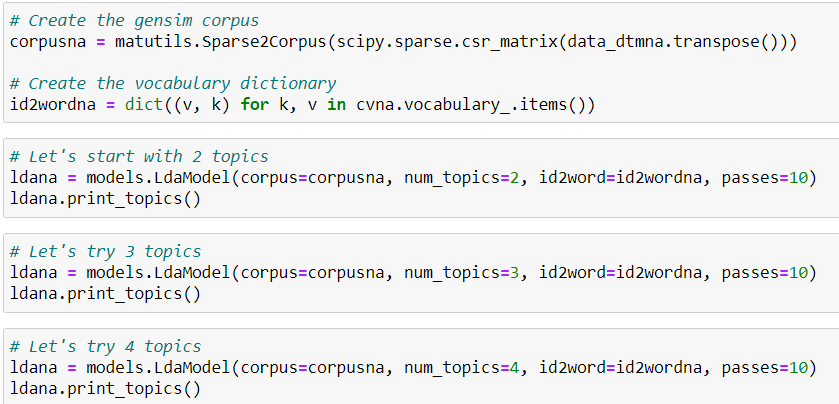


* **Topic Modeling**

For Topic Modeling, we should input our data as a document-term matrix. As each topic will consist of a set of a words where order does not matter, so we are going to use the bag of words format.

We then use ***gensim***, which is a Python toolkit for topic modeling. Gensim has different techniques to do topic modeling, the most popular of which is called **Latent Dirichlet Allocation** (LDA). For this to work, we need to specify the number of topics we think there are in our corpus, and how many iterations we want Gensim to go through our corpus for.

This will output the top words in each topic. It is your job as a human to interpret this and see if the results make sense.

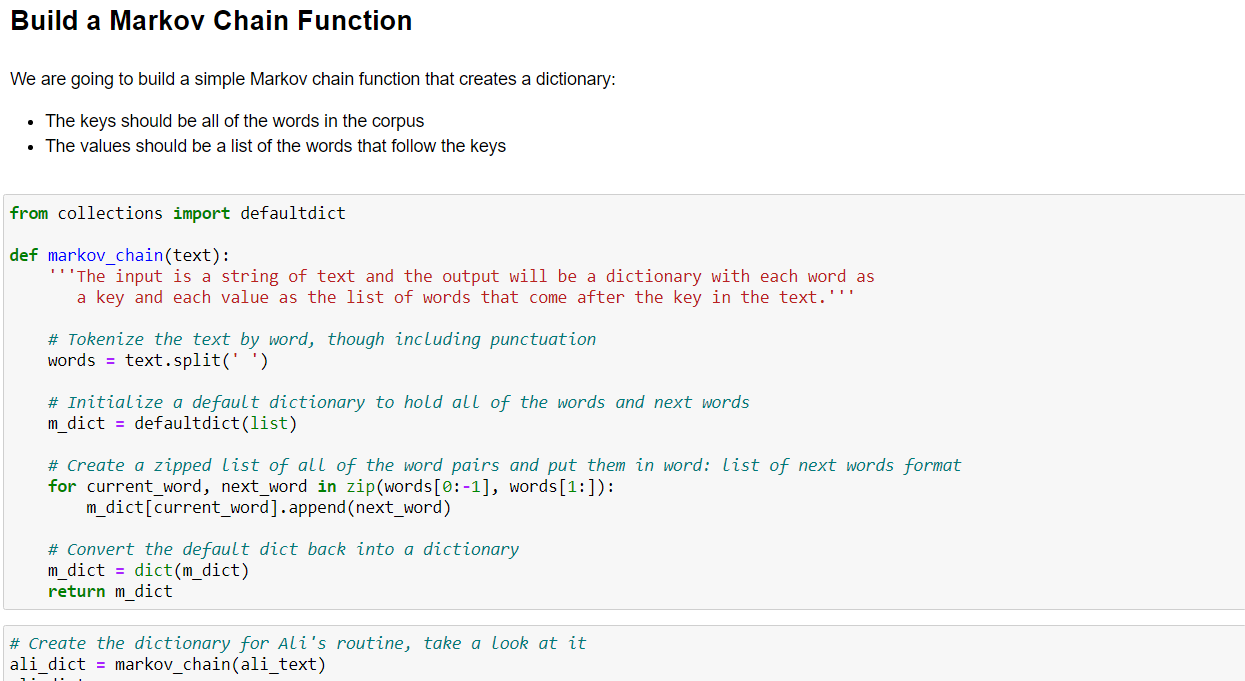
Example:

* **Text Generation**

For Text Generation, we should input our data as a corpus. As we want to preserve the order of the text, including the punctuation.

We can perform this using ***Markov Chains***, which are a way of representing how systems change over time. The main concept behind Markov chains is that they are memoryless, meaning that the next state of a process only depends on the previous state, i.e. It takes every word as a state, and it looks at the next word thinking how likely it is going to be this word based on word number 1.

To apply this, we create a dictionary for a corpus where the keys are the current state and the values are the options for the next state. And we write a function to randomly generate next terms.

Example:

