* The basic flow of data goes in this pipeline of normalization, vectorization, transformation, and then estimation.

**Data Loader & Normalization**

* In my case, the data loader would be the review data from scraping an Amazon product’s site page.
* The collection of reviews is known as a corpus, which is made up of documents. In this case, documents are reviews, which you can think of as a set of strings.
* After collecting our corpus, there are typically a number of preprocessing steps we want to do.
* There are many possible methods you can use like stemming, entity recognition, removing, chunking, and lots of others.
* In my case, I used tokenization, lemmatization, and also removed punctuation.
  + In the context of this project, tokenization is simply breaking up a review by whitespace, into a list of individual words.
  + After tokenization, each word is lemmatized, which means that each word is reduced to their core meaning. A lemma is a word that represents a whole group of words. For example, "I bought these because they're cheap. Use these if you need
  + something done fast, they aren't the greatest." Would be translated into this jumble of words, but it helps keeps consistency in meaning, because we don’t want to differentiate between “purchased, purchasing” or “times, timing, timely, etc.”, because they all are really trying to communicate the same thing, so it’s useful to have this breakdown, but also keeping the context around the words and not removing anything helps keep it grounded.
* Here we assigned a unique integer id to all words appearing in the corpus using a Dictionary class.
* This is analogous to a real-life dictionary in that we build a collection of unique words by sweeping across the texts in the corpus.
* In the end, we see there are 19 distinct words in the processed corpus, which means, in this very small example, the document is represented as a 19-Dimensional vector.
* As you can see here, there are two “be”s in the tokenized list of words, which is only present once in the dictionary.
* So this collection is strings makes up our vocabulary of words that our processing knows about.

**Vectorization**

* In order to perform machine learning on text, we need to transform our documents (or in my case, reviews) into a numeric feature space, because machine learning algorithms only know how to work with numbers, not strings.
* This process is called feature extraction or more simply, vectorization.
* Representing documents numerically gives us the ability to perform meaningful analytics and also creates the instances on which machine learning algorithms operate.

**Bag-Of-Words**

* To convert reviews to vectors, I used a numeric encoding representation called bag-of-words.
* In this representation, each document is represented by a single vector, where each element in the vector represents a question-answer pair, in the style of:
  + Question: How many times does the word ‘bat; appear in this document?
  + Answer: And the answer is, Twice.
* This is why it is advantageous to have a dictionary representation of unique integer ID’s so that we can easily map words to their ID’s.
* This model, while simple, is very effective and forms a starting point for the more complex models I explored.

TFIDF

* While convenient and simple, there are some clear problems with the bag of words model. Two reviews could have the exact same words, but have completely different contexts.
* A better approach would be to consider the relative frequency or rareness of tokens in the document against their frequency in other documents.
* The central insight is that meaning is most likely encoded in the more rare terms from a document. For example, in a corpus of sports text, tokens such as “umpire,” “base,” and “dugout” appear more frequently in documents that discuss baseball, while other tokens that appear frequently throughout the corpus, like “run,” “score,” and “play,” are less important.
* These are just a couple of examples of the many vector spaces you can use. Different models will require different vector spaces, so you will often need to transform from one vector space to another. Some examples are latent semnatic indexing, latent semantic allocation, random projection, and many more.

**Similarity**

* The cosine similarity is described mathematically as the division between the dot product of vectors and the product of the euclidean norms or magnitude of each vector.
* That’s a bunch of magical terms for meaning a similarity measurement taken of the cosine of the angle between the two non-zero vectors Item 1 and Item 2.
* Suppose the angle between the two vectors was 90 degrees. In that case, the cosine similarity will have a value of 0.
* As the cosine similarity measurement gets closer to 1, then the angle between the two vectors A and B is smaller.
* The higher the score, the more similar two documents are.