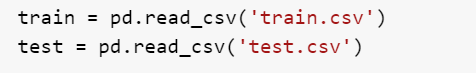
# **Data Preprocessing**

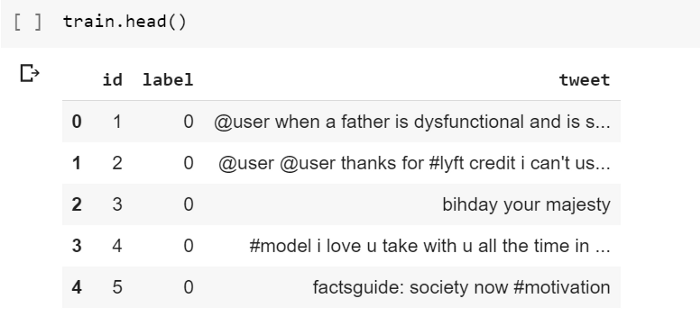
Like always, before we get to the actual nitty-gritty, we need to prepare our data before we can do any of the fun stuff. All of the necessary data is in the GitHub repository.

We have two files, a train file and a test file. Within the train file, we have a random assortment of various tweets. There are three features, a unique id, a label, and the actual tweet text. The label is ‘0’ if the tweet is non-offensive (not racist or sexist, etc.) and the label is ‘1’ if the tweet is very offensive.

First, we’ll import a whole bunch of libraries.

Now we will use pandas to read the test and train files.



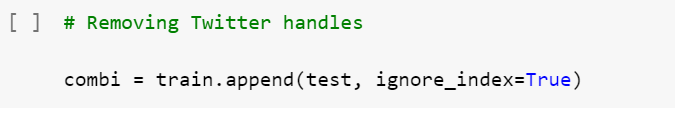
Let’s use pandas to actually see what our data looks like.

# **Data Cleaning**

For data cleaning, we’ll take three steps:

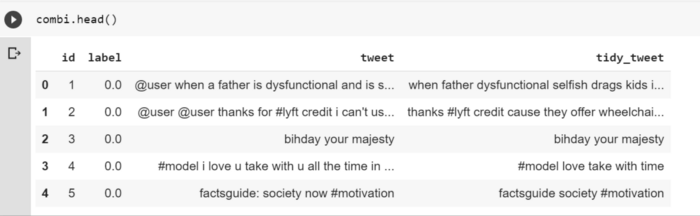
1. Remove Twitter handles
2. Remove punctuation, numbers, and special characters
3. Remove smaller words that don’t add much value

In order to save time, we’ll combine our train and test dataframes and do all the data cleaning on both.



Now we’ll create a regular expression that allows us to pinpoint and remove the Twitter handles from our tweets.

|  | def remove\_pattern(input\_txt, pattern): |
| --- | --- |
|  | r = re.findall(pattern, input\_txt) |
|  | for i in r: |
|  | input\_txt = re.sub(i, '', input\_txt) |
|  |  |
|  | return input\_txt |
|  |  |
|  | combi['tidy\_tweet'] = np.vectorize(remove\_pattern)(combi['tweet'], "@[\w]\*")  Now our combined dataframe should no longer have Twitter handles. We can remove the special characters, numbers, and punctuation with a simple script.  Then we’ll remove the short words that add no value.  Let’s see how our combined dataframe looks like now. |

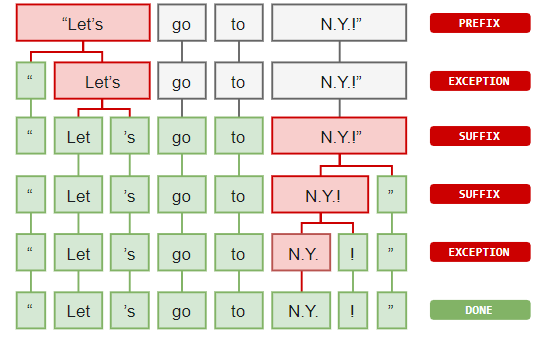


# **Tokenization**

Here’s where the real fun begins! Because machine learning algorithms can’t find patterns and insights like us (they can’t just read in a whole bunch of text so they have to take in individual words), we need a way to preserve the *“essence”* of each tweet while making it as easy as possible for the algorithms to learn what makes a tweet offensive or not.

Thankfully, we have this useful form of pre-processing called **tokenization**, which is basically splitting up all of your text into the bare-bones units.

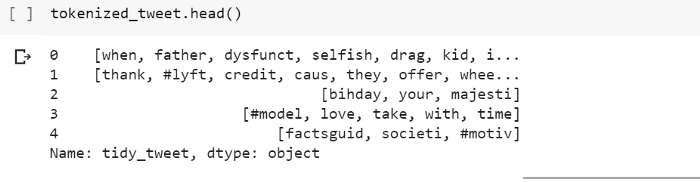
Here’s an example of what that would look like.



Tokenizing is actually really easy. All you have to do is split up your text into the individual components.

# **Stemming**

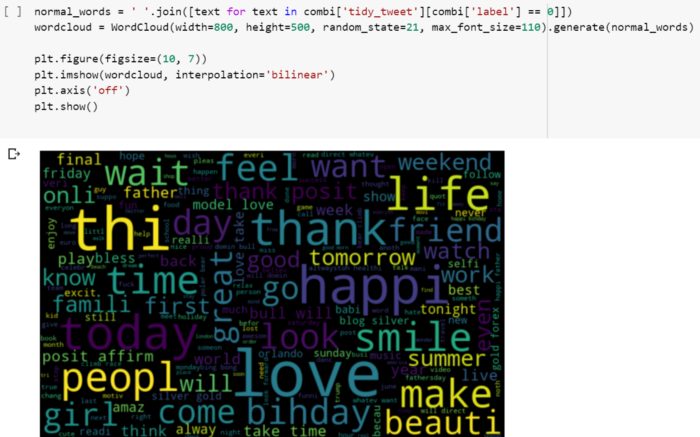
Now that we’re done tokenizing, we need to move on to a further pre-processing step called **stemming**, which is essentially converting each tokenized word into its root form and tense. For example, stemming the word “creating” would result in the word “create” and the word “consignment” would become “consign”



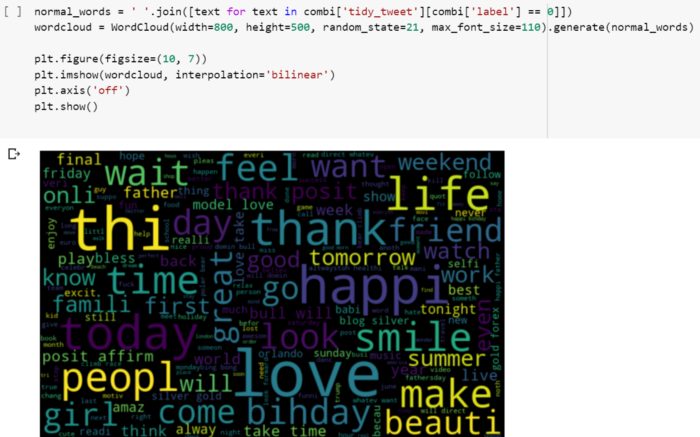
# **Data Visualization**

Before we move on to the machine learning, it’s a good practice to get some intuition into what’s going on in our data. Data visualization is a great way for us to actually explore and understand our data.

## **Common Words**

To see the common words in our tweets, we can create a wordcloud to actually SEE them.

## **Words in Positive Tweets**



## **Words in Negative Tweets**

*(Warning: the words in this wordcloud are definitely NSFW)*



## **Hashtags**

We purposely didn’t remove the hashtags because they can carry a ton of useful information. We’ll extract the hashtags and see which ones come up most often.

# **Extracting Features**

Now that we’ve done all the necessary steps, it’s time to actually get to the machine learning.

We will be using the logistic regression machine learning algorithm to train our prediction model. Before we can actually train our model, we have to transform our text data into a format the algorithm can actually read.

There are two methods we can use, Bag-of-words and TF-IDF (term frequency–inverse document frequency). I won’t go too much into detail in how these feature extraction models actually work, but feel free to learn more about them through the resources I’ve placed at the bottom of this article.

First we’ll extract with bag-of-words.

**Model Building**

Time to build our models.

First we’ll build with the bag-of-words dataframe.

|  | from sklearn.linear\_model import LogisticRegression |
| --- | --- |
|  | from sklearn.model\_selection import train\_test\_split |
|  | from sklearn.metrics import f1\_score |
|  |  |
|  | train\_bow = bow[:31962, :] |
|  | test\_bow = bow[31962:, :] |
|  |  |
|  | xtrain\_bow, xvalid\_bow, ytrain, yvalid = train\_test\_split(train\_bow, train['label'], random\_state=42, test\_size=0.3) |
|  |  |
|  | lreg = LogisticRegression() |
|  | lreg.fit(xtrain\_bow, ytrain) |
|  |  |
|  | prediction = lreg.predict\_proba(xvalid\_bow) |
|  | prediction\_int = prediction[:,1] >= 0.3 |
|  | prediction\_int = prediction\_int.astype(np.int) |

Now let’s see the results of our prediction accuracy.

