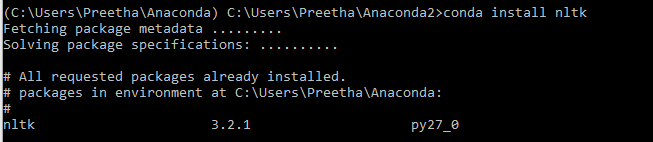
**PYTHON CODE FOR IDENTIFYING THE GENDER OF A PERSON BASED ON FIRST NAME**

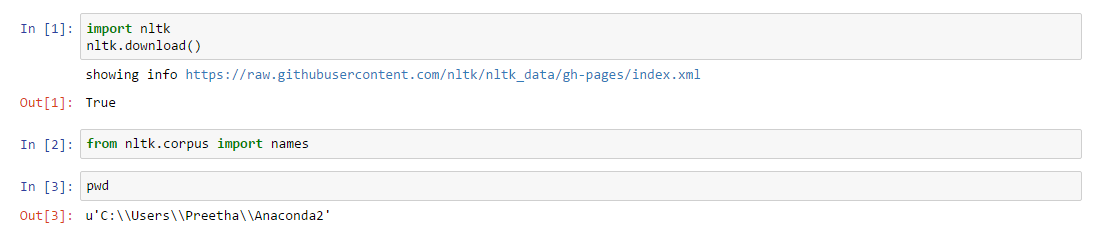
The following approach is based on the usage of the Names corpus in the Natural Language Tool Kit (NLTK), to train a Naïve Bayes classifier to identify the gender of a person, on the basis of the first name. The names corpus consists of 8000 names catagorized by gender. The classification labels are ‘male’ and ‘female’ in this case. The male names were stored in a text file called male.txt. The female names were stored in a text file called female.txt.

All codes were written in a Jupyter notebook for readability purposes. Screen shots from the notebook have been attached. Python 2.7 was used in this coding exercise.

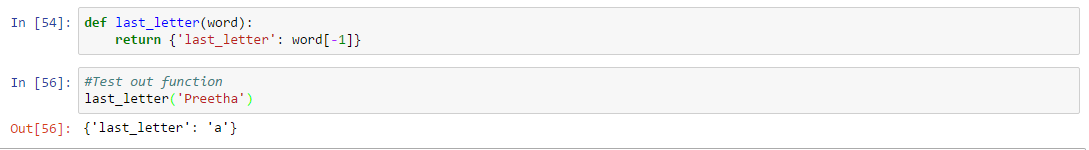
Make sure the NLTK package was installed. Check Anaconda command prompt:



Import the Names corpus:



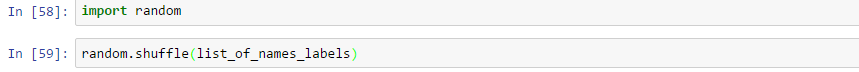
It has been observed that female names tend to end in vowels and male names tend to end in consonants. This information was used in the process of defining a function called last\_letter, where the objective was to extract the last letter of a first name. Please note that since first names are in the form of strings, it is possible to have negative indexes for strings. In this context, the negative index of -1, is applied to the function last\_letter to extract the last alphabet of a person’s first name.



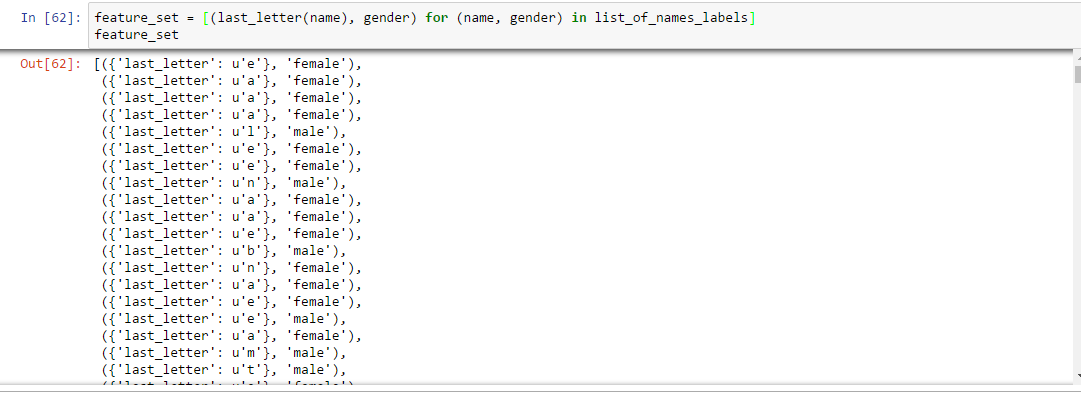
The next step is to extract all the examples contained in the names corpus (the first name and corresponding gender label) and assign it to an object called list\_of \_names\_labels. These are the examples classified by gender, meant for the training set and testing set:



The next step is to randomly shuffle the labelled examples using the shuffle() method from the random library:



Next, the function last\_letter is applied to derive what is known as a feature set. This feature set is nothing but a dictionary that essentially classifies first names into either male or female on the basis of the last letter extracted from each of the 8000 first names in the names corpus!



Next, split 8000 examples in the proportion of 70:30 – please note that 30% of the data was kept in the training set, while 70% of the data was kept in the testing set. This strategy did achieve a better accuracy score! There was also little divergence between the accuracy score of the training set and testing set, as a result of this approach!



NLTK has a built in Naïve Bayes classifier

Reference 1: <http://www.nltk.org/_modules/nltk/classify/naivebayes.html>

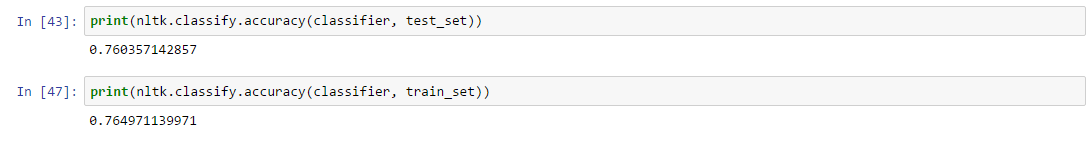
Reference 2: <http://www.nltk.org/howto/classify.html>

I next, put the classifier to the test, by testing it out on first names that were not present in the training set. Here, NLTK’s classify package was used. The object called classifier (created in the previous step) was applied to NLTK’s classify package and the function created in the earlier step called last\_letter, was called on classifier.classify and last\_letter took as its input a string in the form of a first name. The job of the classify package here is to assign a class label to a token, that takes the form of a character string (which in this case is the first name of a person):



Reference: <http://www.nltk.org/api/nltk.classify.html>

I next, took a look at the accuracy scores of the classifier for both the training set and testing set:



**Reference for the entire case study**

# Bird, S., Klein, E. and Loper, E. (2009) *Natural Language Processing with Python,* O'Reilly Media