##### Convert English Text from written expressions into spoken forms

# Abstract

The purpose of the work reported here is to take some initial steps towards addressing deficiencies in previous approaches to text normalization. In English language picking up words like currency, date and time, local expression to make them fluent and accurate in Machine Learning. It is important factor to make it proficient at Machin level.

Many speech and language applications, including text-to-speech synthesis (TTS) and automatic speech recognition (ASR), require text to be converted from written expressions into appropriate "spoken" forms. This is a process known as text normalization, and helps convert 11:52 to "eleven fifty-two" and $3.16 into "three dollars, sixteen cents”.

The main task of this project is to automate the process of developing text normalization grammars with Machine Learning and various Algorithms.

# Data Collection and description

We have chosen our data from the <https://www.kaggle.com/>.

It contains a large corpus of text. Each sentence has a **sentence\_id**. Each token within a sentence has a **token\_id**. The **before** column contains the raw text, the **after** column contains the normalized text, the training set contains an additional column, **class**, which is provided to show the token type. Like PLAIN, DATE, PUNCATION, CARDINAL, ELECTRONIC, MEASURE, DIGIT, FRACTION, MONEY, ADDRESS. In addition, there is an **id** column used in the submission format. This is formed by concatenating the **sentence\_id** and **token\_id** with an underscore (e.g. 123\_5).

* en\_sample\_submission.csv - file showing the correct format
* en\_test.csv - the test set, does not contain the normalized text
* en\_train.csv - the training set, contains the normalized text

# Algorithm and model

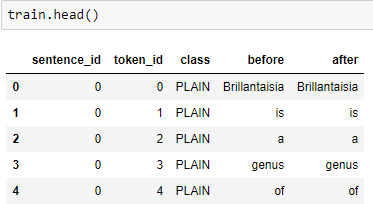
For the given data we are going to apply different algorithms to make text format into appropriate spoken forms. By analysing data and further processing. Hence, we can predict the after words that can be obtained by processing the before words from the data.

We are implementing Machine Learning models like Logistics and Multinomial Naïve Bayes. We have also implemented Lesk algorithm.

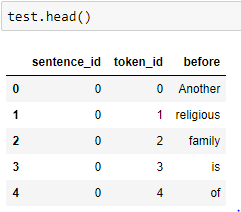
# Data preprocessing

Before making the data model suitable for further processing, we have implemented following Exploratory Data Analysis.

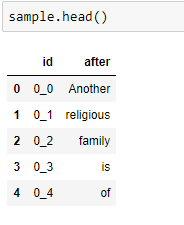
##### Train Dataset



##### Test Dataset

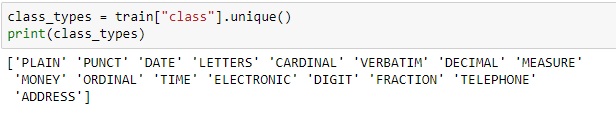


##### Sample Dataset



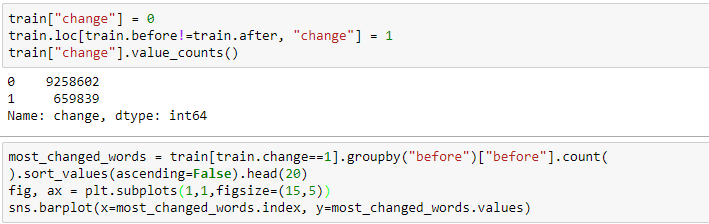
##### Unique Class Type

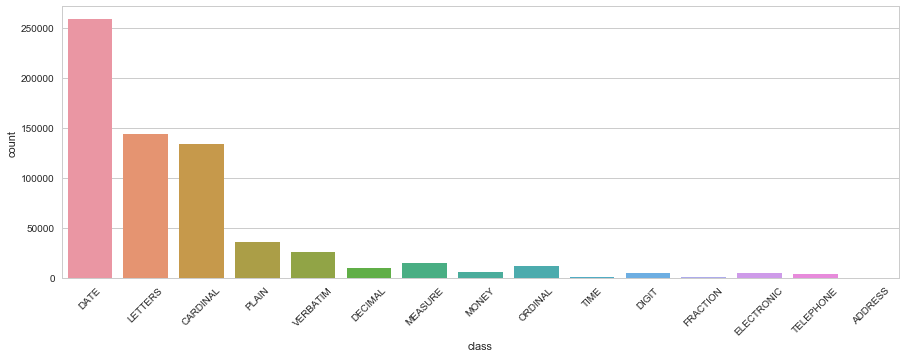
The given dataset contains different class type data. Let us check the class types.



##### Most Changed word from class type

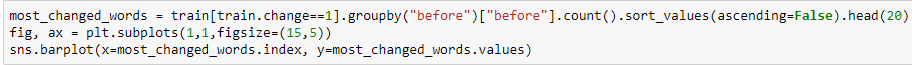
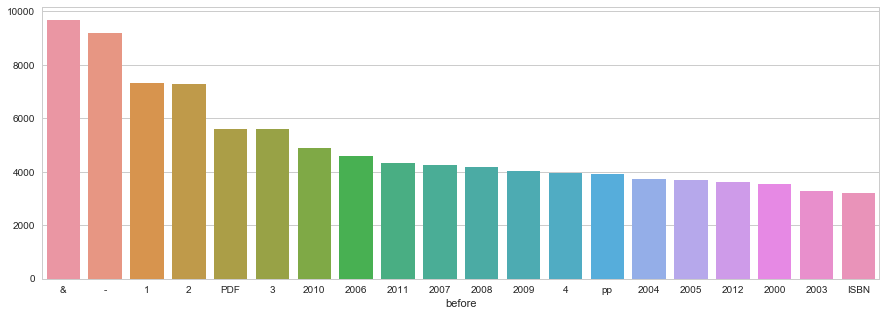
In order, to identify the most changed class type, we first create a column name as “change” and set the value to “0”. We set the change value to 1 for difference between before and after text.





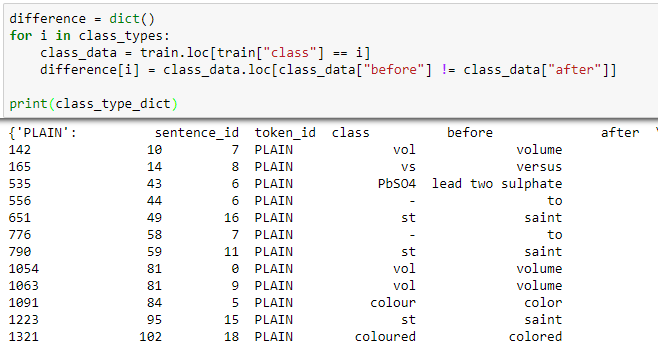
##### Top 20 Most Frequently Changed word

Here we query for only the changed dataset and group it based on before text. We sort the result in descending order and get the top 20 words from the dataset.

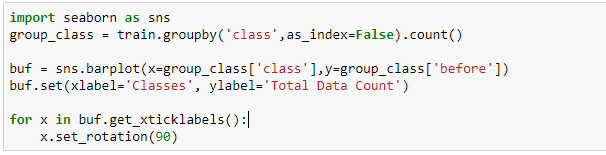
 

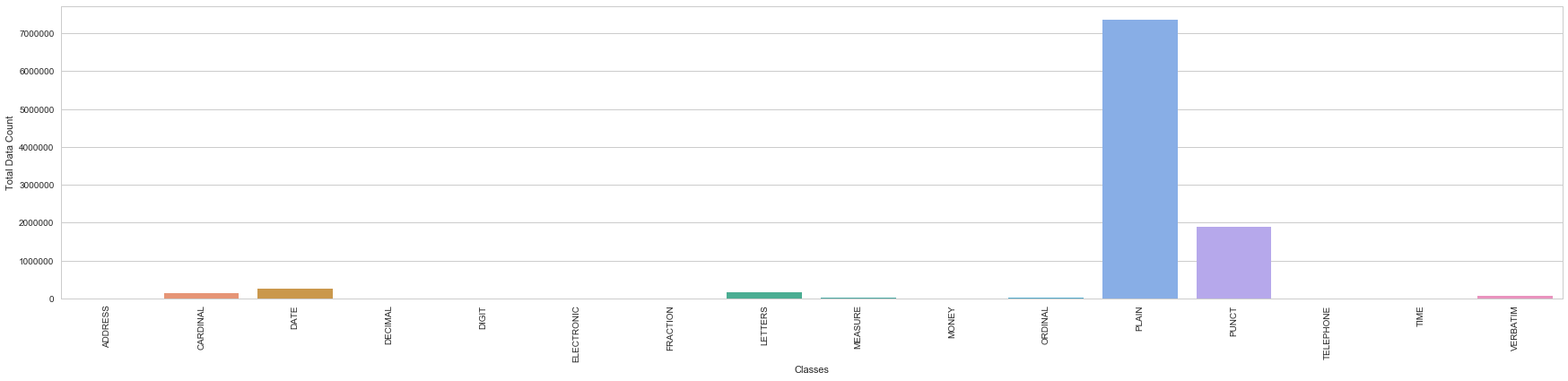
##### class based Dictionary

According to class type, the dictionary stores the changed words in the dataset.

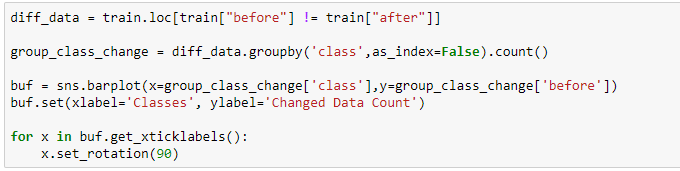


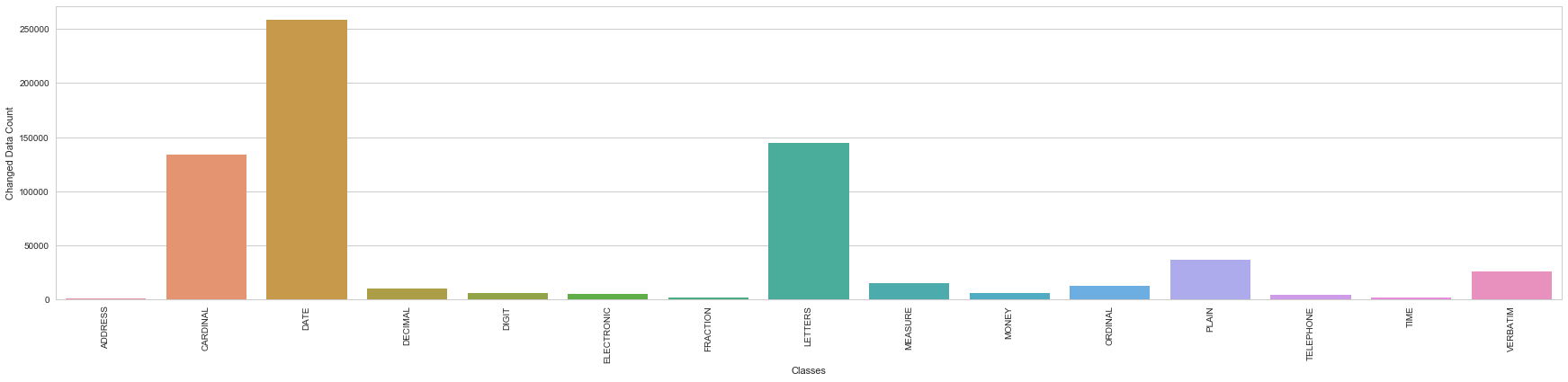
##### Data analysis Before column





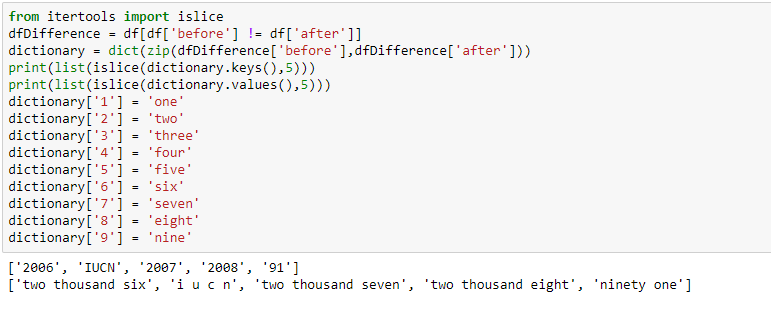
##### Data Analysis after column





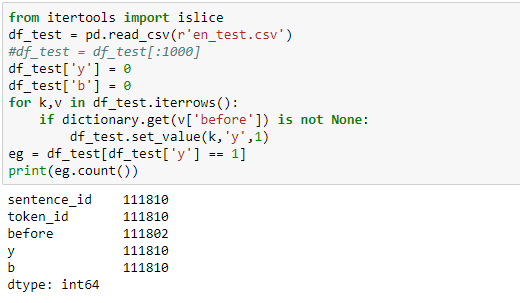
# Creating a dictionary

We created a dictionary, to store the values that are changed in the dataset. A dictionary will help us identify the changed values from before to after.

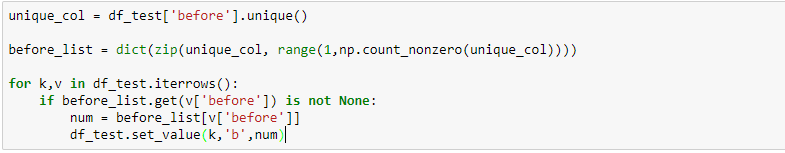


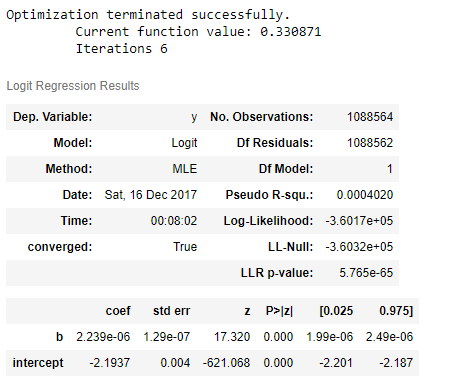
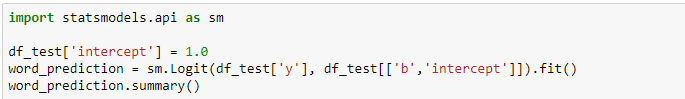
##### identify words in test dataset present in the dictionary

Previously, we created a dictionary to store the values of changed word. Now we identify the words present in the test dataset using this dictionary. If the word is found then we set the flag to 1.



##### Create a unique number for the words in the test dataset

Here we created a unique number for the words present in the test dataset.

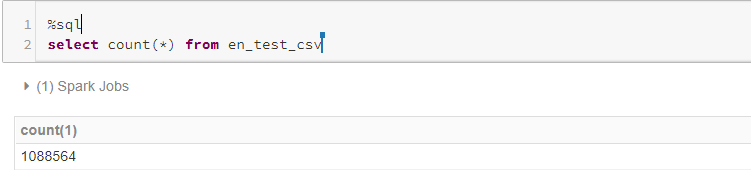


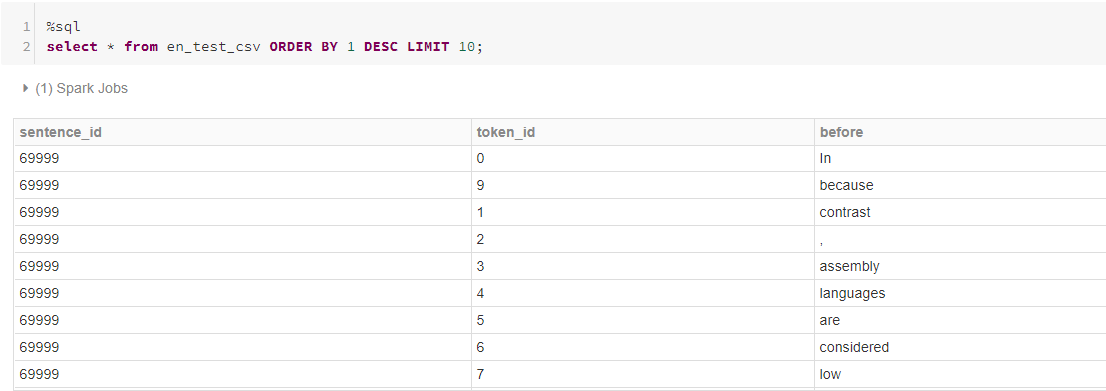
# SQL Queries for Data Analysis

We uploaded the dataset to data bricks for performing SQL queries. We wrote a script to upload all the csv files of dataset to data bricks using REST API.

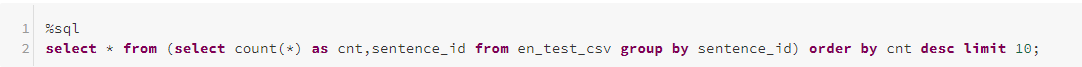


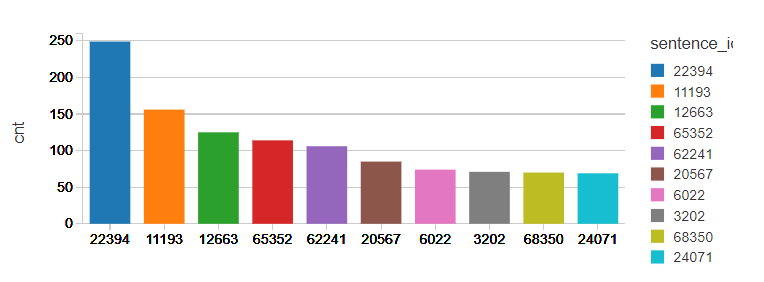
After this, we created a database for storing the dataset into the tables. Below we performed various analytical queries.



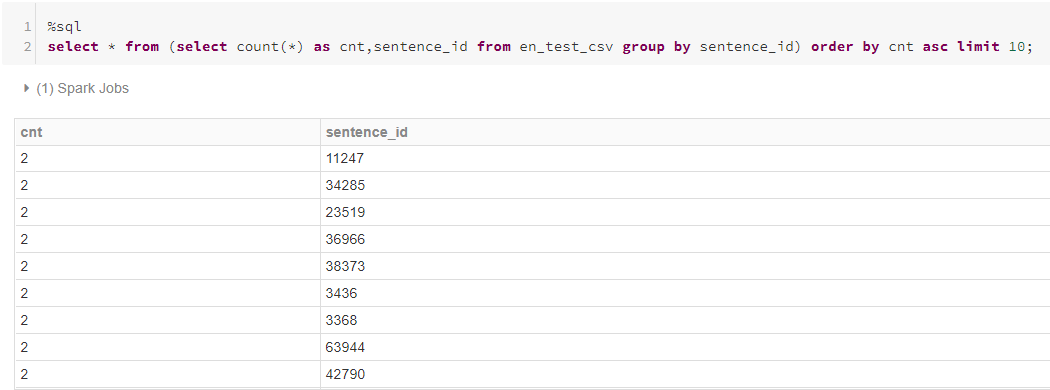


##### Top 10 longest sentences in the test dataset

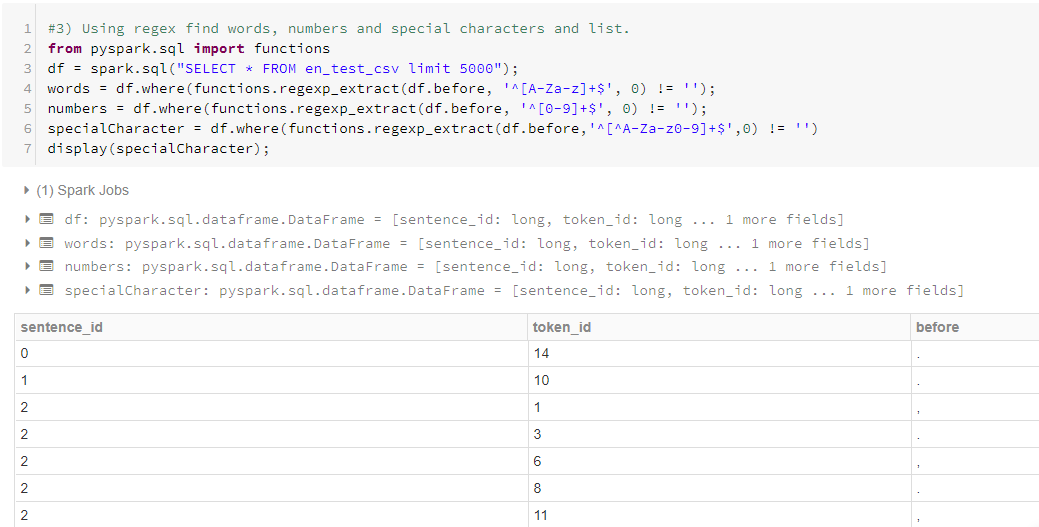




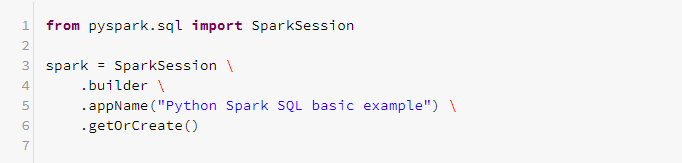
##### Top 10 shortest sentence in the test dataset

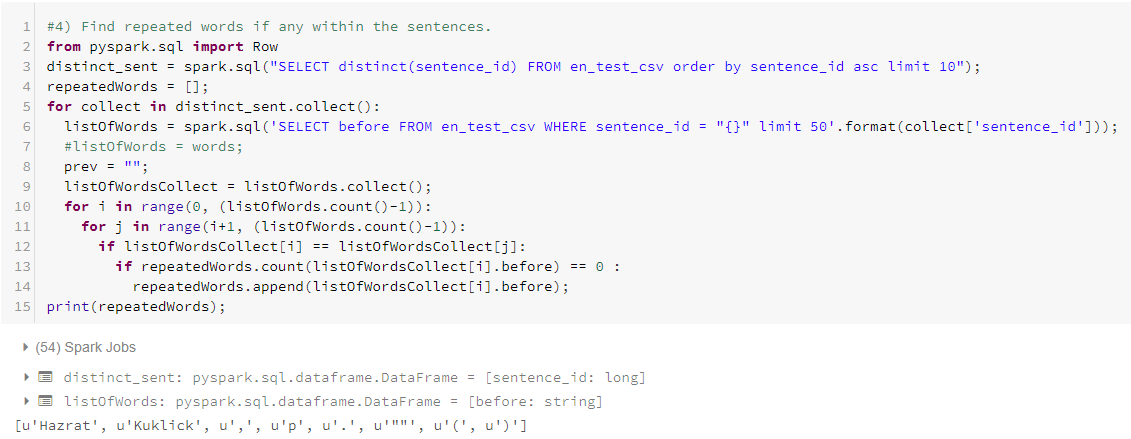


##### Additional query to find words numbers and special characters

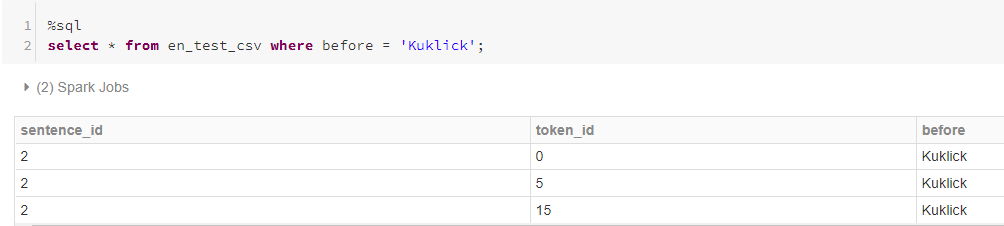


##### Find repeated words within a sentence

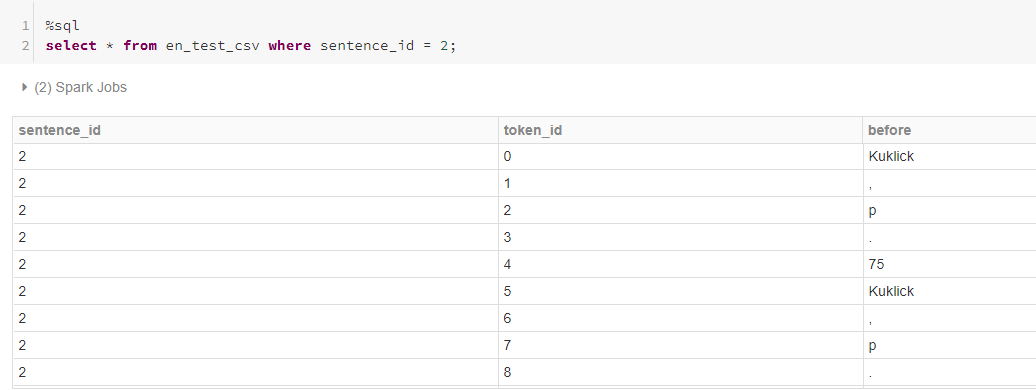




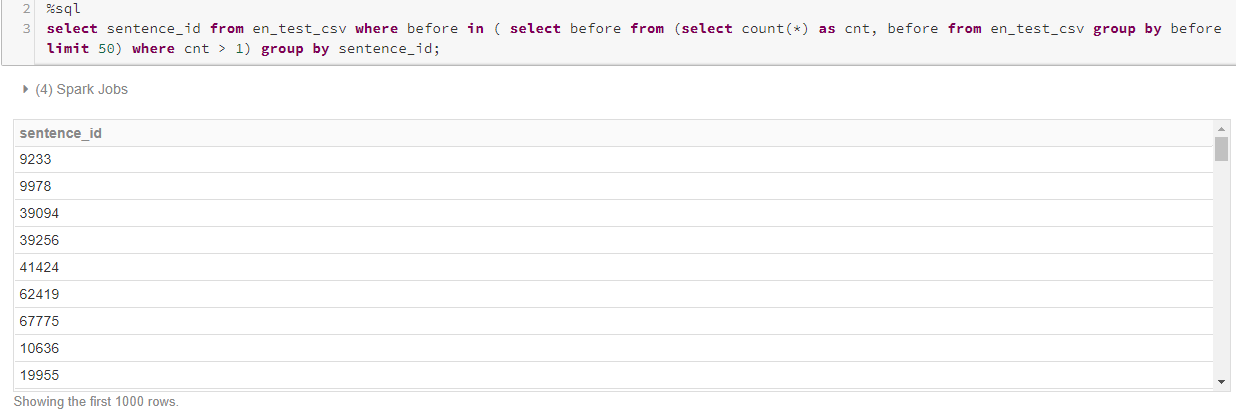
##### Query to fetch a word that is repeated in the same sentence.



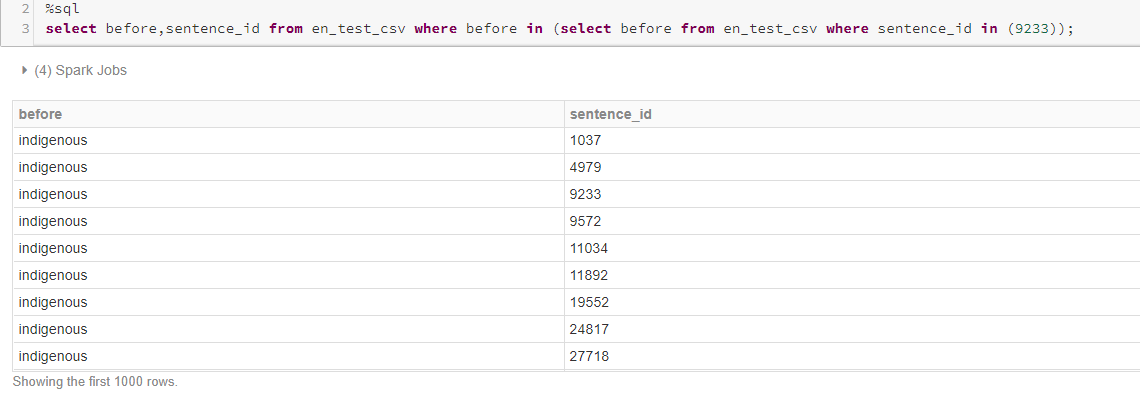
##### Query to display the whole sentence which contains repeated word like “Kuklick”.



##### A word that contains in more than one sentences.



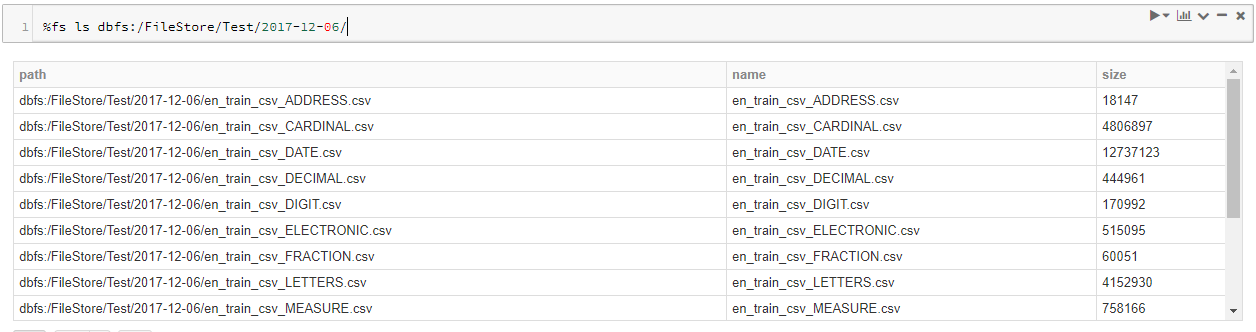
##### Query to display the words that is present in more than one sentence.

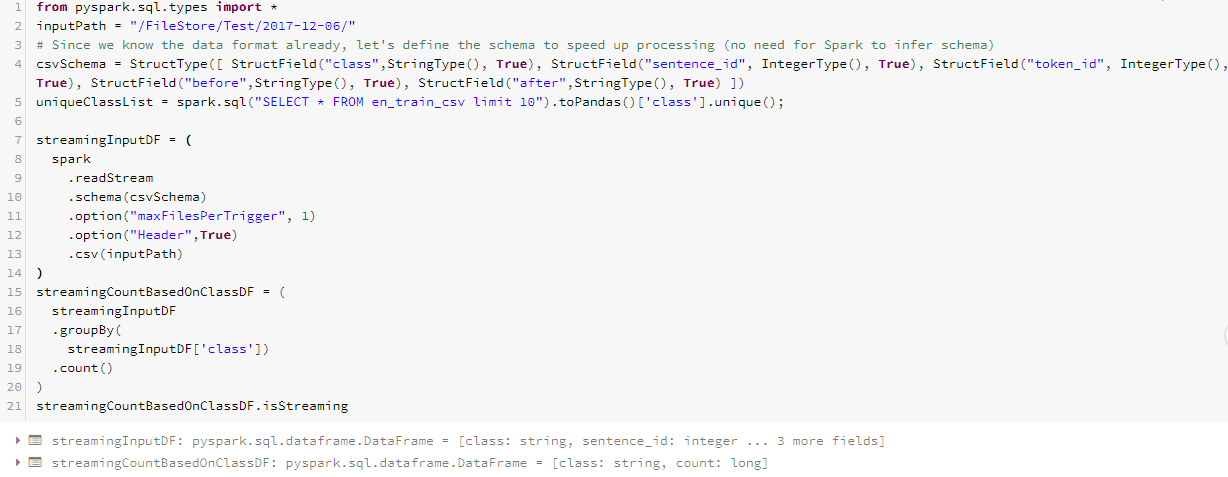


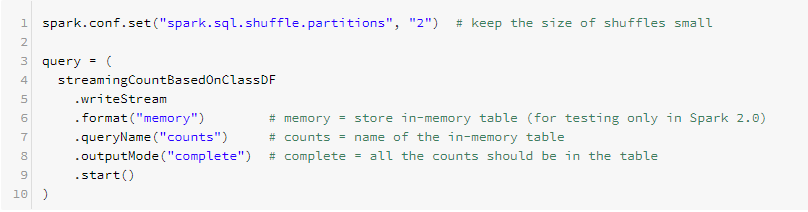
# Structured Streaming

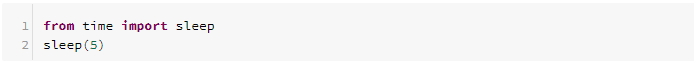
##### Created multiple files according to the class type



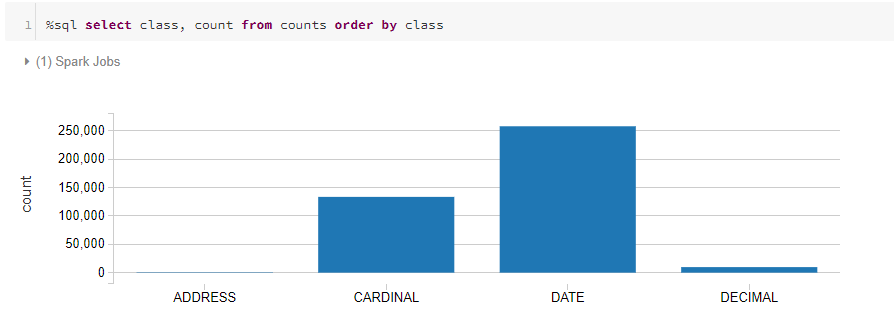




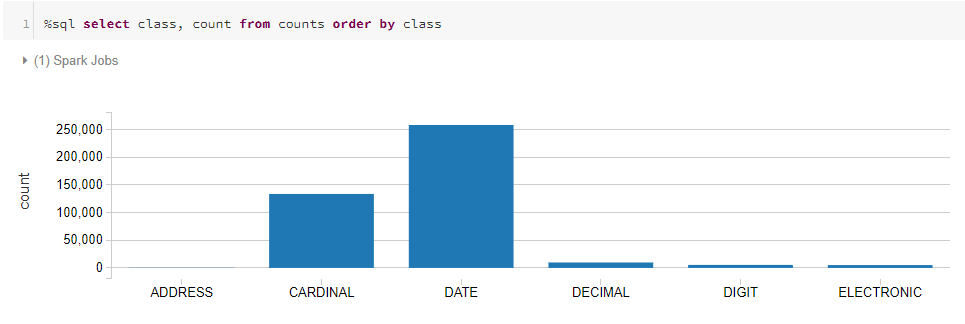


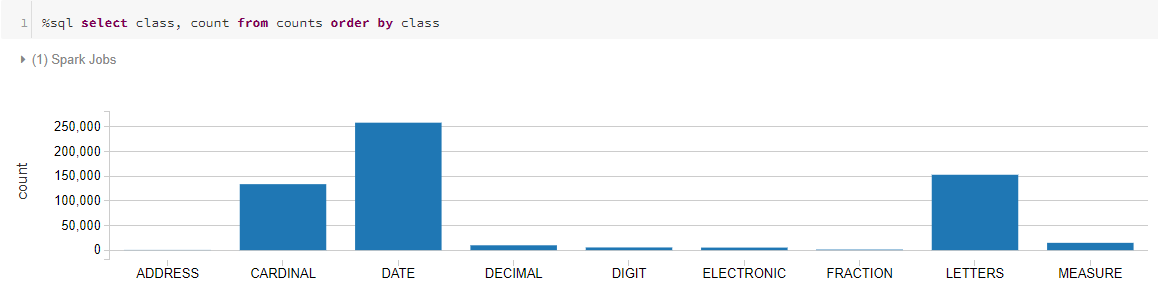


##### Below is the streaming analysis of dataset.







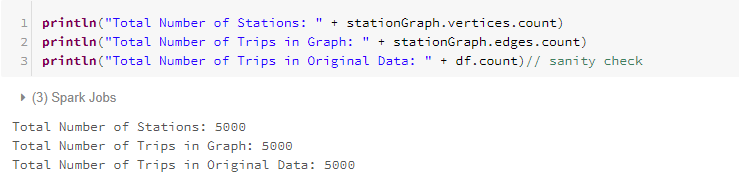


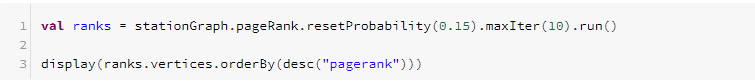
# GraphFrame using Scala

For Graphframe analysis, we used scala since GraphFrames only work on it. We set the vertices as before values and created edges between before and after values.

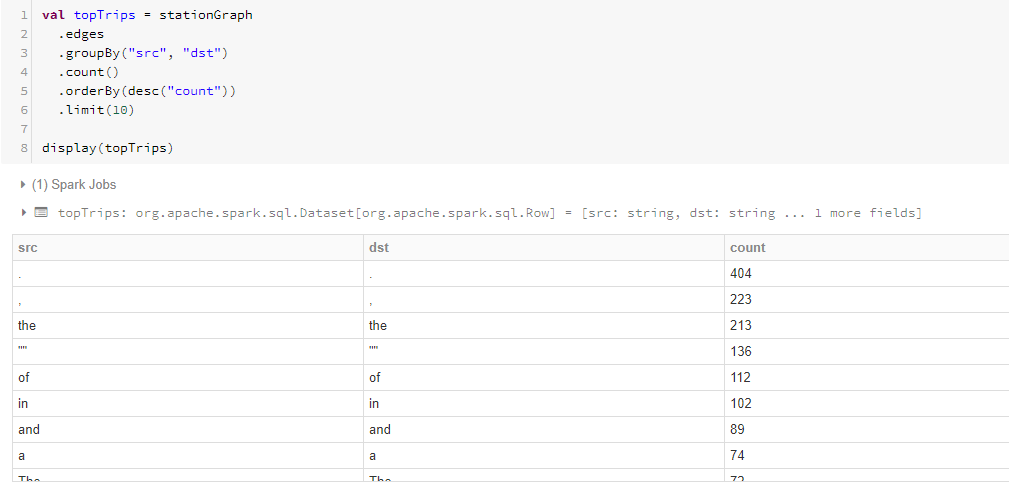


##### We queried to fetch the total number of vertices and edges.





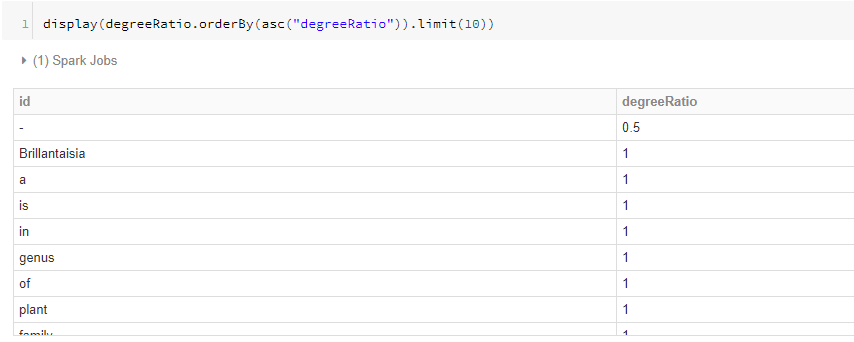
##### Total number of trips from source(‘Before’) to destination(‘After’) based on the edges.



##### Maximum times a words changed from before to after.

##### 

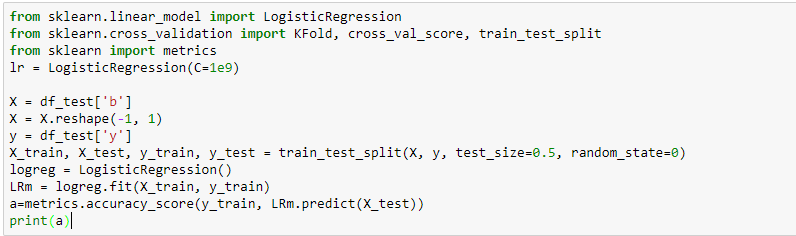
##### Below are the values that show the most frequently change words



# Predicting the accuracy

##### LogisticRegression

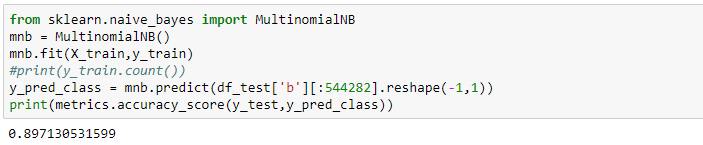
We used logistic regression to identify the accuracy for the test dataset. We have successfully got 89%.



0.897442869689

We used multinomial naïve Bayes to check for better results.

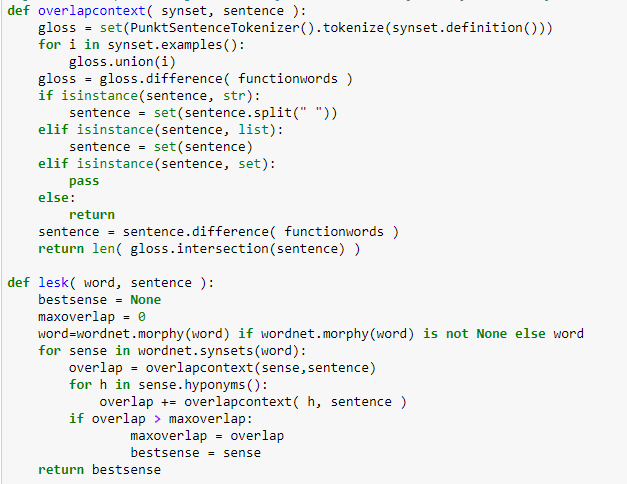
##### multinomial naive bayes



We got similar results for Multinomial Naïve Bayes.

##### Lesk Algorithm

Assumption in this algorithm, is that neighboring words would share similar context. Below functions identifies the glossary of a word and compare with the sentences. If there are similar neighboring words then it predicts best sense of the word in question. We tested this algorithm against out dataset and found that it is not meeting our requirement and this lead to more research into RNN approaches to text normalization.



# Applications

* Social Media; twitter, facebook
* Google text to speech engine
* Mobile applications like google talkback, iOS speech

# Future Scope

After implementing the Lesk algorithm, we found the research can progress towards RNN approach of Text Normalization which is a research paper presented by Richard Sproat, Navdeep Jaitly.

# References

<https://en.wikipedia.org/wiki/Text_normalization>

<https://www.kaggle.com>

<https://arxiv.org/ftp/arxiv/papers/1611/1611.00068.pdf>

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

<https://en.wikipedia.org/wiki/Lesk_algorithm>

https://github.com/Akirato/Lesk-Algorithm