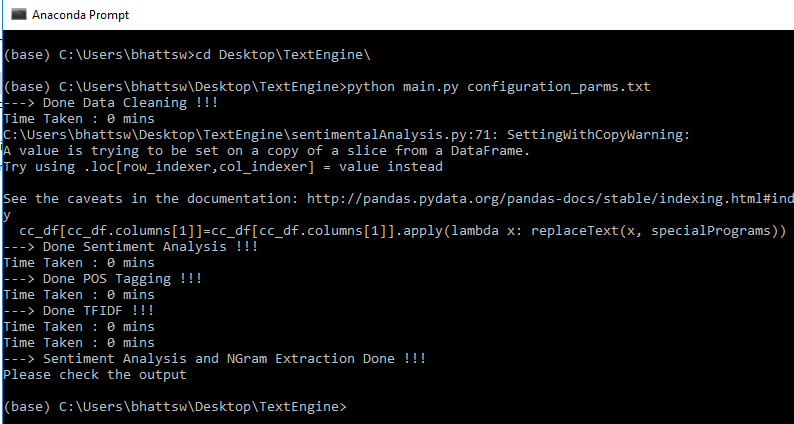
**Text Engine Documentation**

The text engine performs two major tasks on the given data – classifies the text into a ‘query’ or a ‘complain’ through sentiment analysis, and extracts n-grams from the text.

The text engine consists of 3 python modules: main.py, dataCleaning.py, sentimentAnalysis.py, posTagging.py, tfidf.py, NgramExtraction\_category.py and a configuration file: configuration\_params.txt.

**Running the text engine:**

To run the text engine run main.py in the terminal giving the path of the configuration file along with it.



**configuration\_params.txt:**

inputDataFormat (Mandatory): file - to be entered when the input is a file

repoDirectoryPath: Mention the path where the delivered repositories are placed.

outputDataFormat: Output data format - either file/hive is to be mentioned here.

runSentiment = Enter 'y' if you want Sentiment analysis from your input data.

runNgram: Enter 'y' if you want Ngrams generated from your input data.

outputFilesLocation: Output files location path; if selected 'file' for 17th element

topicOnGrams: Please enter 1 / 2 / 3 - to get topic modelling on either unigrams, bigrams or trigrams.

generateUniqueID: Auto generate Unique ID for you, this option is to be selected only when there is no unique identifier for your data. Enter (y) if this has to be enabled. Please make sure that text is the first column in this case.

groupByID : Enter 'y' if you are having multiple unique identifiers and want to group by identifier like MTN Number. Please note that, you will lose all other filter columns.

inputFileNameWithPath: Absolute location with file name to be provided if 'file' is mentioned in the option 3.

inputDataSeparator: if, 'file' is mentioned in option 3... example: \t - for tab delimited file.

**main.py :**

This module takes in configuration parameters from the configuration file and imports other python modules to be used by the test engine in order to call appropriate functions to read and clean the data, and, run sentiment analysis and n-gram extraction by passing the data along with other required parameters.

**dataCleaning.py:**

Reads and preprocesses the data, and generates unique IDs for each entry if not present (which is specified by a configuration parameter). This function sets the unique Id column as the first column of the inputted data frame and text as the second column. It also groups the data according to the Ids and removes duplicate entries. The text of the data frame is then preprocessed to ease NLP operations.

Function called from main: fnReadData()

Parameters passed: inputDataFormat, inputDataSeparator, inputFileNameWithPathinputHeader, generateUniqueID, groupByID

Files needed: data in a csv format

**sentimentAnalysis.py:**

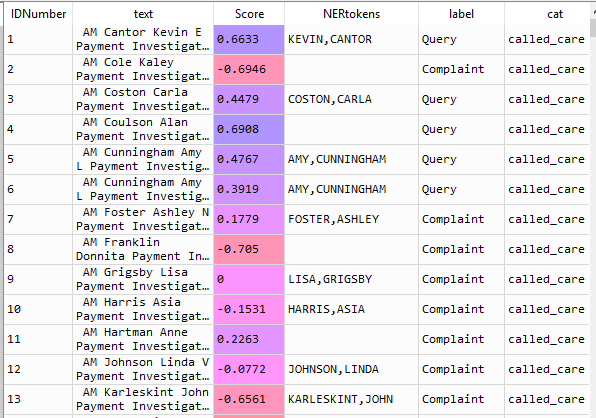
Reads biased tokens, Named Entity Recognition organization names, and special programs from the repository and calculates a sentiment score for each entry, along with a label (either complain or query). This function also makes a list of NER tokens present in each entry. This information is stored in a csv file format.

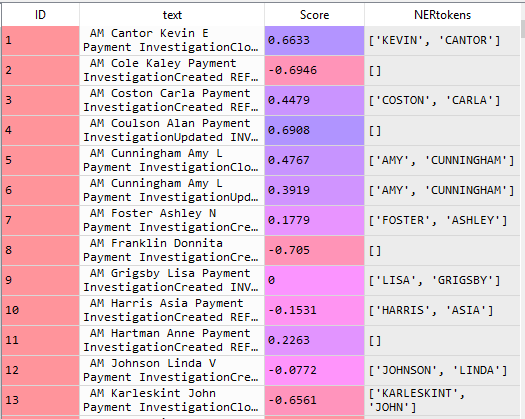
Function called from main: fnSentimentAnalyzer()

Parameters passed: cc\_df, repoDirectoryPath, folderToWrite, cc\_df1, outputDataFormat, runSentiment, prodRun

Files needed in repository: biasedTokens.txt, nerOrganisations.txt, specialPrograms.txt

Joined\_data - Data frame saved in csv format



cc\_df - Data frame returned by the function back to main.py

**posTagging.py**

The text data is vast and certain techniques need to be applied to only take relevant data out from the text, to carry out NLP operations more efficiently. Some of such techniques applied are – lemmatization, removing bogus words, removing irrelevant characters (e.g. – ‘\*’, ‘.’ etc.), removing uni-length tokens and also filtering text using parts of speech tags (nouns, adjectives adverbs etc.).

**tfidf.py**

Creates a data frame to be used as input for n-gram extraction. The text column is preprocessed and returned.

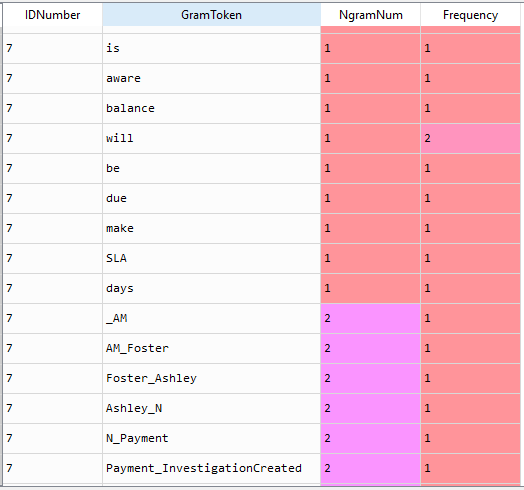
**NgramExtraction\_category.py:**

Finds uni-grams, bi-grams and tri-grams for each entry; removes the bogus bigrams read from a text file and calculate the N-gram count along with the frequency of each and store the result in a csv file format. In the resulting data frame each entry has a uni/bi/tri-gram along with the n-gram count (1/2/3) and its frequency in the given entry (i.e. frequency within the same query or complain that has the same ID number).

Function called from main: fnExtractNgrams()

Parameters passed: folderToWrite, transformed\_ce, outputDataFormat, repoDirectoryPath, topicOnGrams, runNgram, prodRun

Files needed in repository: bogusBigrams.txt

OP1\_df - Data frame saved in csv format

cc\_df\_topic\_grams - Data frame returned by the function back to main.py (topicOnGrams=3 in configuration files)

