

Bansilal Ramnath Agarwal Charitable Trust’s

# Vishwakarma Institute of Information Technology

Business Intelligence and Data Analytics

(Assignments 3)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Roll No. | GR No. | Batch |
| Chetan Chinchulkar | 324017 | 21810350 | E9 (Comp D) |
| Nikit Gokhe | 324022 | 21810522 | E9 (Comp D) |
| Sahil Langha | 324032 | 21810533 | E9 (Comp D) |
| Kunal Parkhe | 324040 | 21810049 | E9 (Comp D) |

GROUP MEMBERS:

# Assignment No. 03

# Aim:

1. Perform text preprocessing with creation of inverted index for unstructured data (text). Consider suitable data set. **(Use Python/R )**

B. Study and implement  opinion mining / sentiment analysis for sample online/offline application     **(Use Rapid Miner)**

# Theory:

Term Frequency (TF):

 Suppose we have a set of English text documents and wish to rank which

Document is most relevant to the query, “Data Science is awesome!”

 A simple way to start out is by eliminating documents that do not contain all three

words “Data”,”is”, “Science”, and “awesome”, but this still leaves many documents.

To further distinguish them, we might count the number of times each term

Occurs in each document;

 The number of times a term occurs in a document is called its term

frequency.

The weight of a term that occurs in a document is simply proportional

to the term frequency.

Formula:

tf(t,d)=count of t in d /number of words in d

Document Frequency:

 This measures the importance of document in whole set of corpus, this is

very similar to TF.

 The only difference is that TF is frequency counter for a term in

document d,whereas DF is the count of occurrences of term in the

document set N.

 In other words,DF is the number of documents in which the word is

present. We consider one occurrence if the term consists in the

document at least once, we do not need to know the number of times

the term is present.

Formula : df(t) = occurrence of t in documents

Inverse Document Frequency(IDF):

 While computing TF, all terms are considered equally important.

However it is known that certain terms, such as “is”, “of”, and “that”,

may appear a lot of times but have little importance.

 Thus we need to weigh down the frequent terms while scale up the

rare ones, by computing IDF, an inverse document frequency factor

is incorporated which diminishes the weight of terms that occur very

frequently in the document set and increases the weight of terms that

occur rarely.

 IDF is the inverse of the document frequency which measures the

In formativeness of term t. When we calculate IDF, it will be very low for

The most occurring words such as stop words (because stop words such

as “is” is present in almost all of the documents, and N/df will give a

very low value to that word). This finally gives what we want, a relative

weightage.

Formula : idf(t) = log(N/(df + 1))

TF-IDF Formula :

tf-idf(t, d) = tf(t, d) \* log(N/(df + 1))

# Python code :

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

def computeTFIDF(tfBagOfWords, idfs):

    tfidf = {}

    for word, val in tfBagOfWords.items():

        tfidf[word] = val \* idfs[word]

    return tfidf

def computeIDF(documents):

    import math

    N = len(documents)

    idfDict = dict.fromkeys(documents[0].keys(),0)

    for document in documents:

        for word,val in document.items():

            if val>0:

                idfDict[word] +=1

    for word,val in idfDict.items():

        idfDict[word] = math.log(N/float(val))

    return idfDict

def computeTF(wordDict,bagOfWords):

    tfDict = {}

    bagOfWordsCount = len(bagOfWords)

    for word,count in wordDict.items():

        tfDict[word] = count / float(bagOfWordsCount)

    return tfDict

documentA = 'WE study in VIIT,India'

documentB = 'I live in India and I am proud of my country'

bagOfWordsA = documentA.split()

bagOfWordsB = documentB.split()

uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))

numOfWordsA = dict.fromkeys(uniqueWords,0)

for word in bagOfWordsA:

    numOfWordsA[word] +=1

numOfWordsB = dict.fromkeys(uniqueWords,0)

for word in bagOfWordsB:

    numOfWordsB[word] +=1

tfA = computeTF(numOfWordsA,bagOfWordsA)

tfB = computeTF(numOfWordsB,bagOfWordsB)

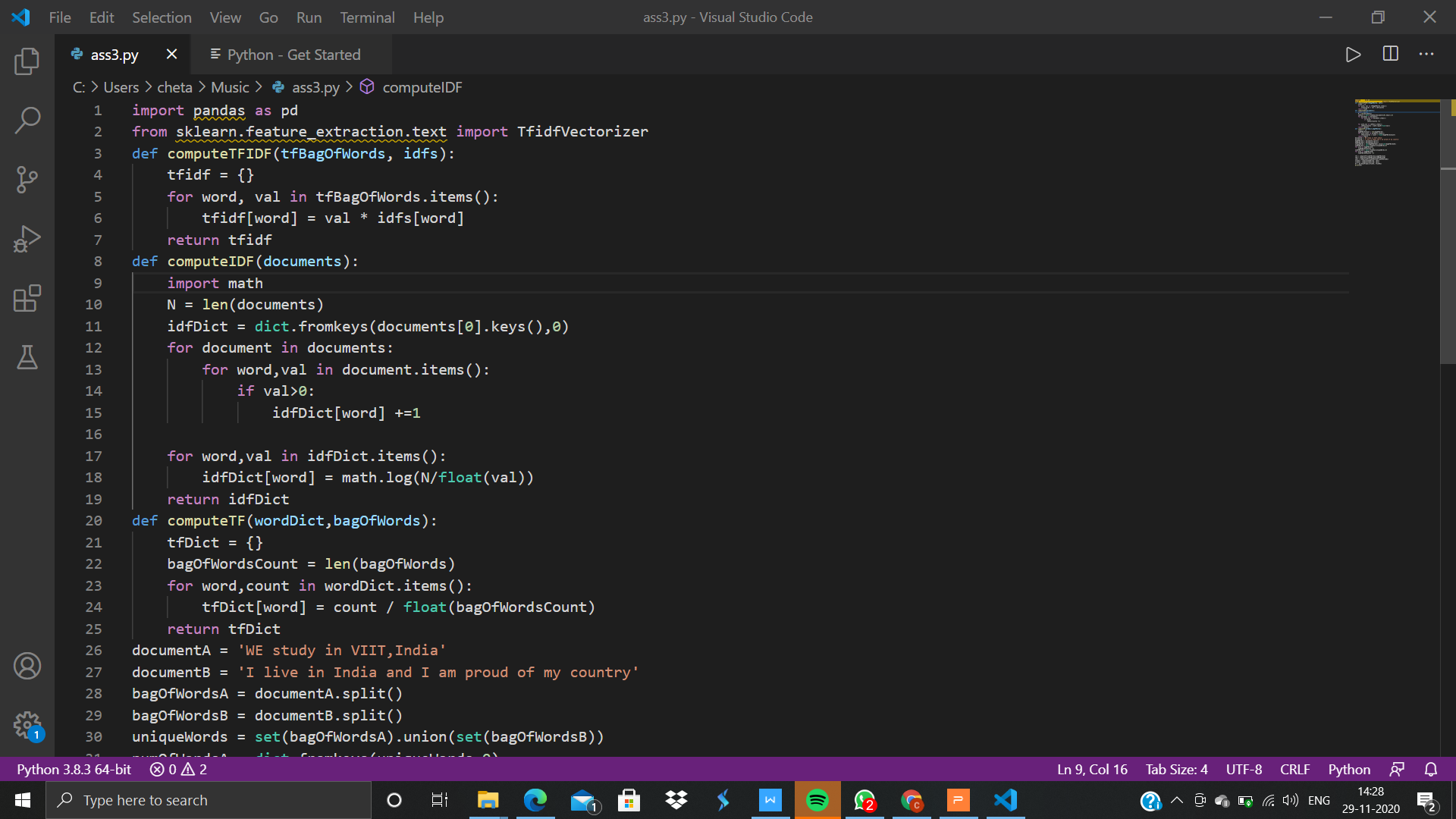
idfs = computeIDF([numOfWordsA,numOfWordsB])

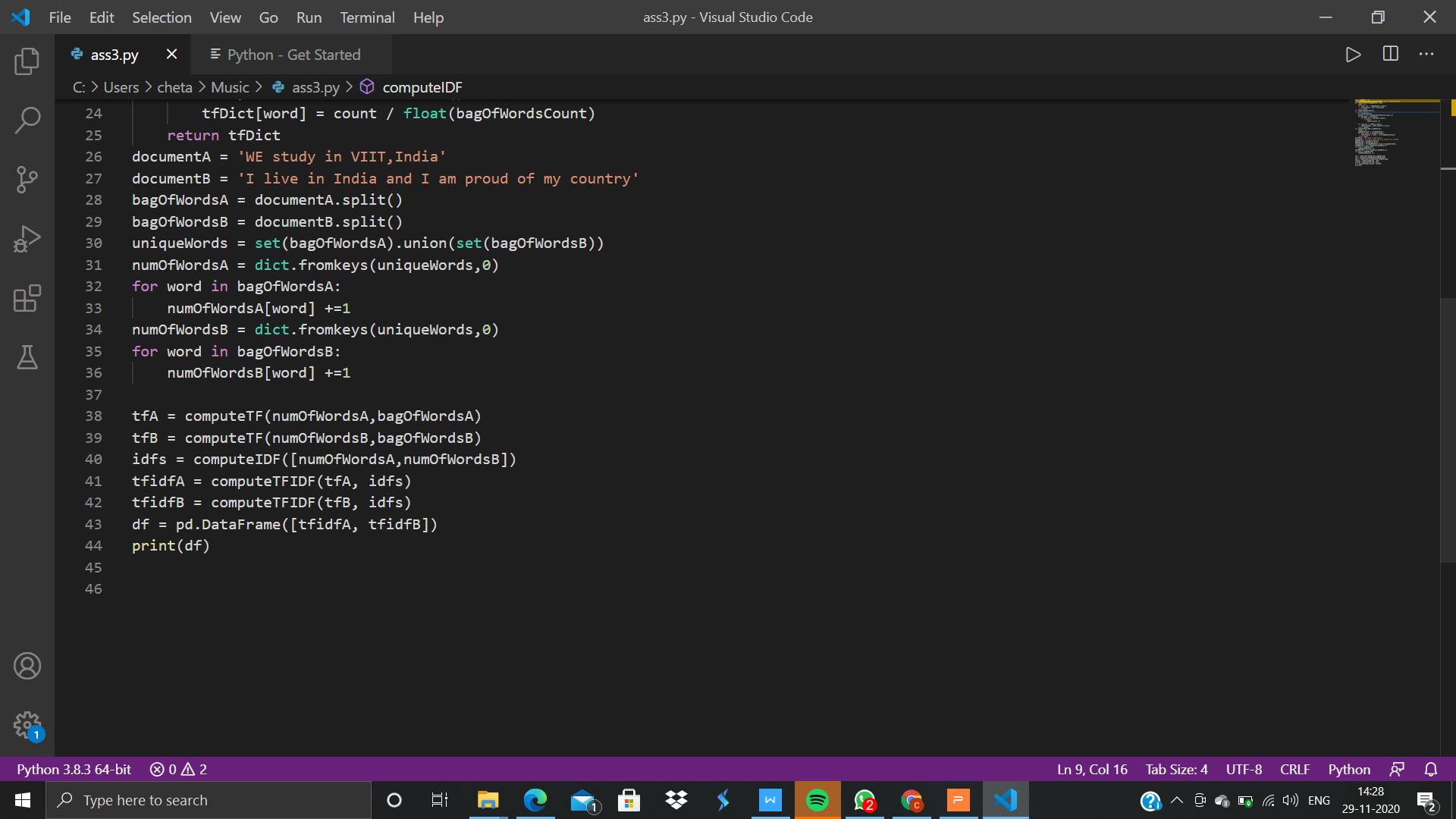
tfidfA = computeTFIDF(tfA, idfs)

tfidfB = computeTFIDF(tfB, idfs)

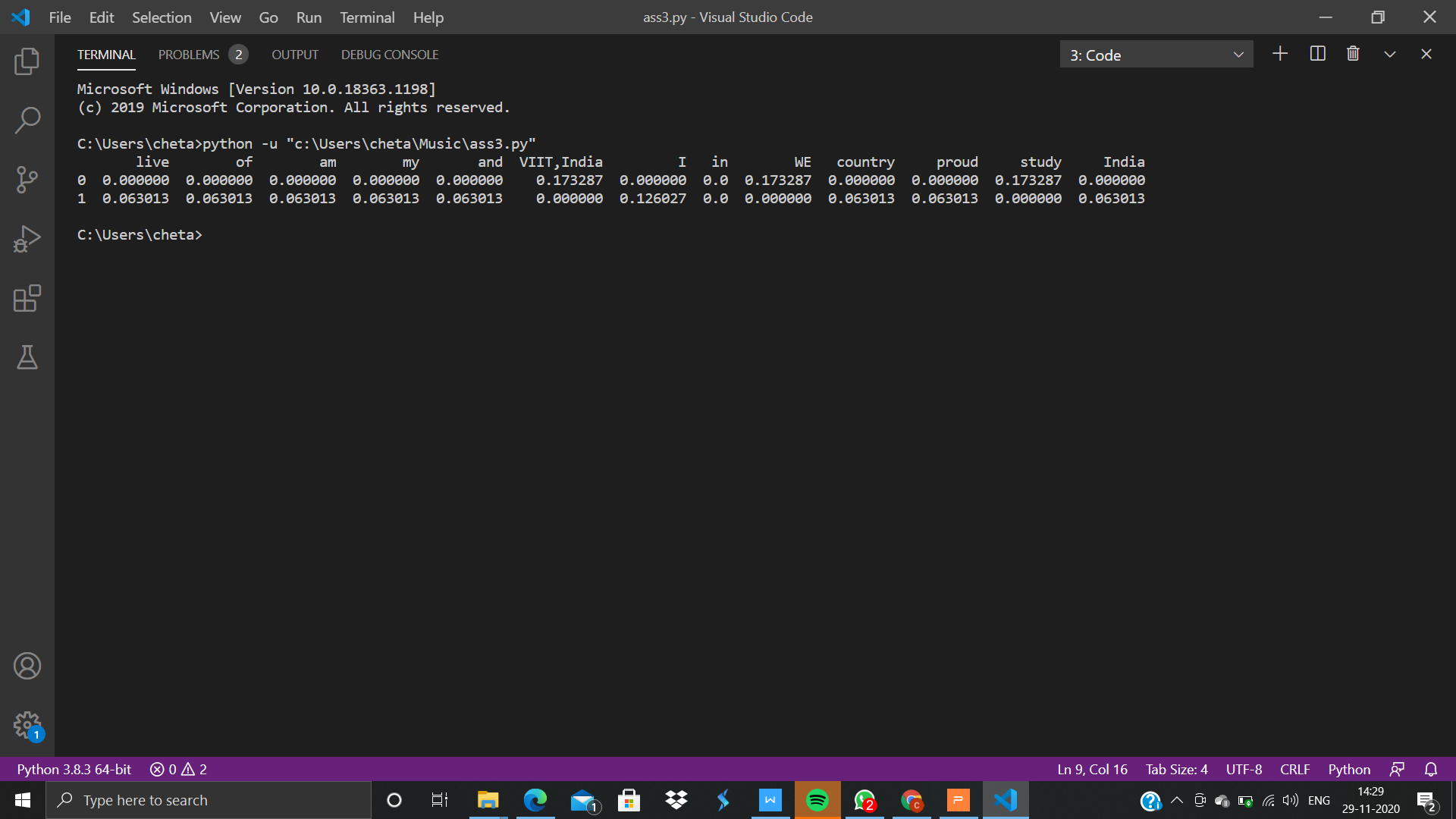
df = pd.DataFrame([tfidfA, tfidfB])

print(df)





# Output:



# Conclusion:

1. Successfully calculated Tf-Idf for taken data set of stories.

2. Tf-Idf is a great way to measure the rank of the document for a given

corpse.

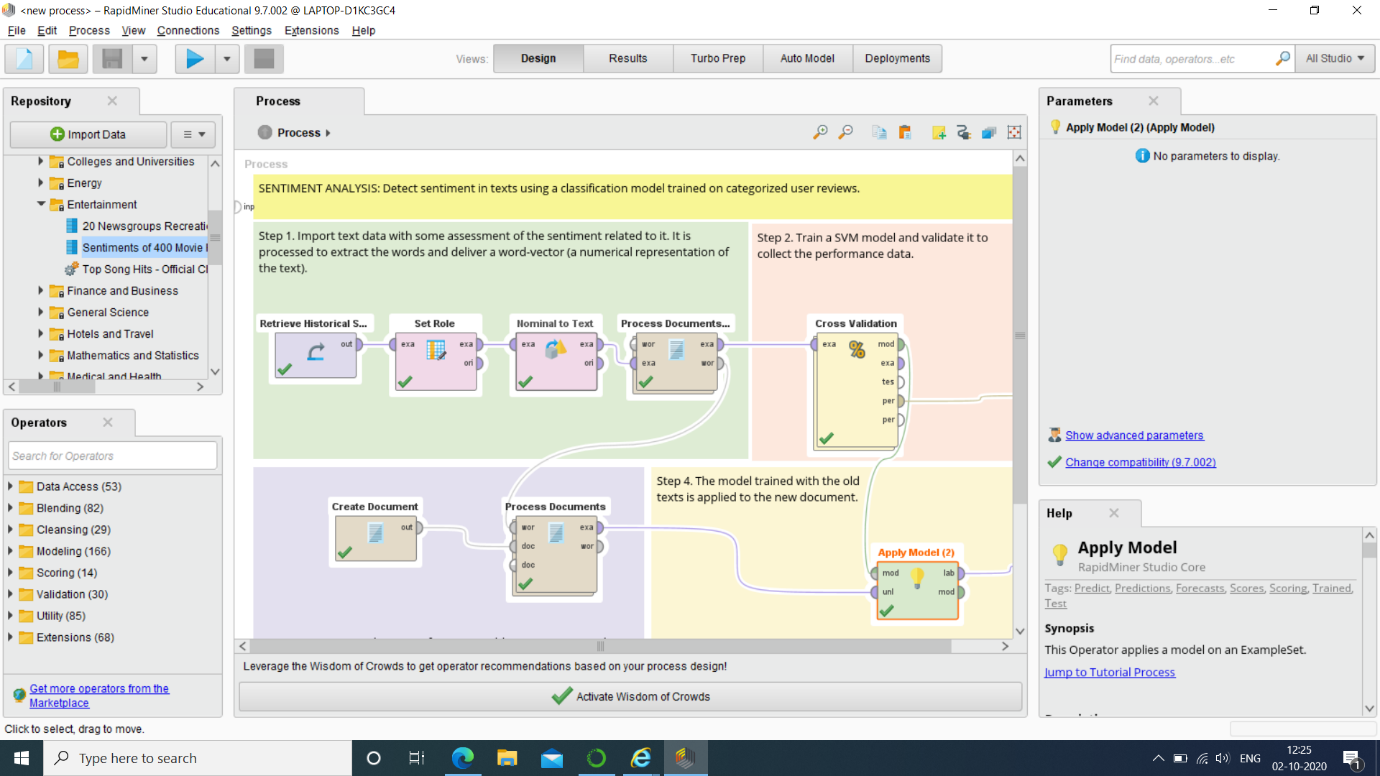
3. Python provides a great and easier tool to calculate Tf-Idf for the given

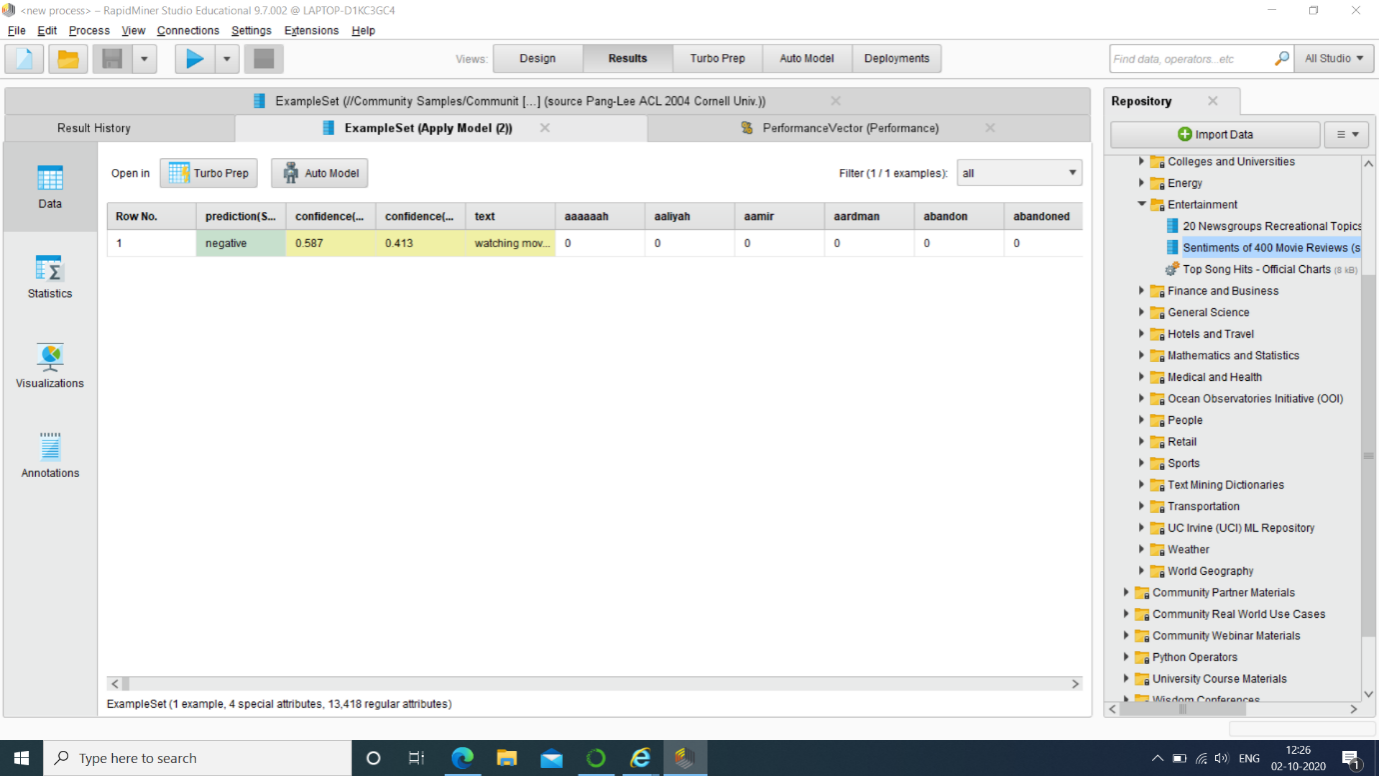
corpse.

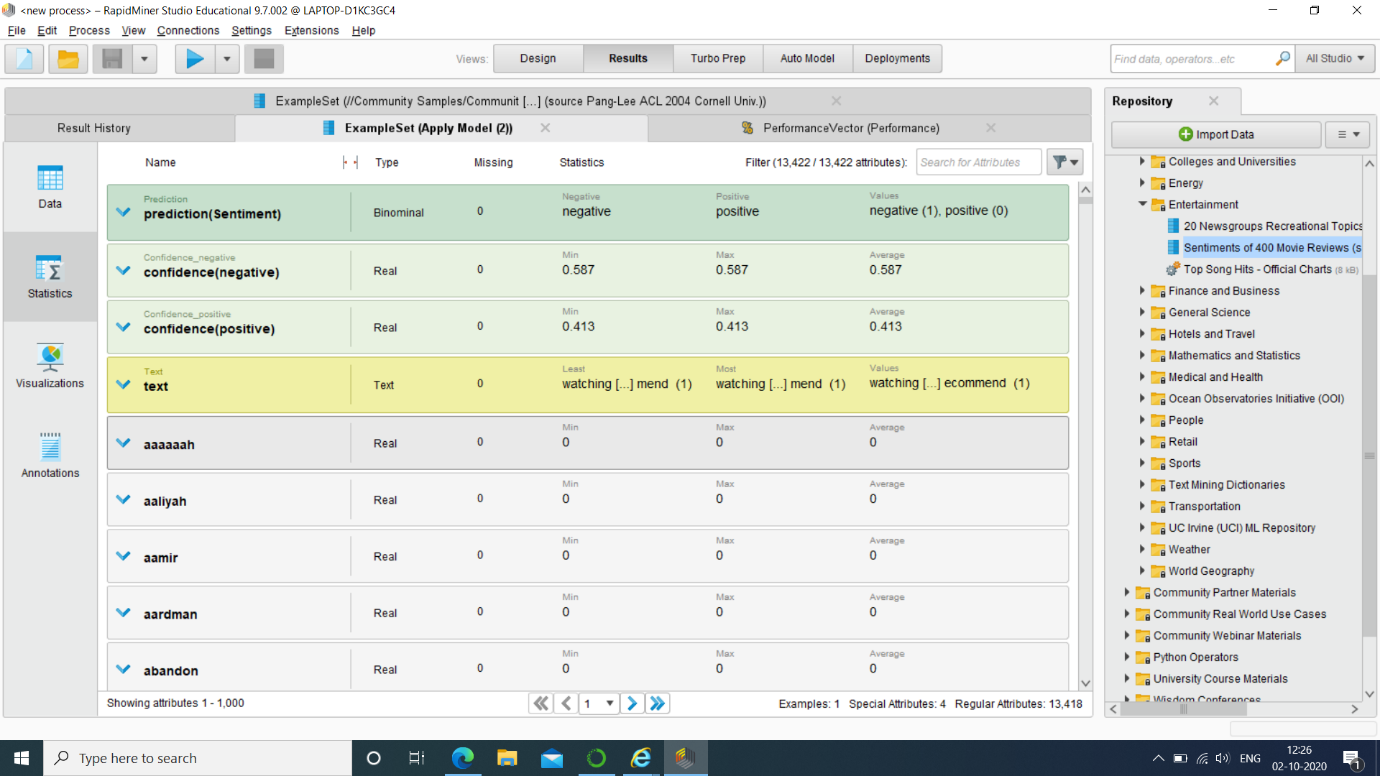
# Text Analysis of Movie reviews using Rapid Mainer:

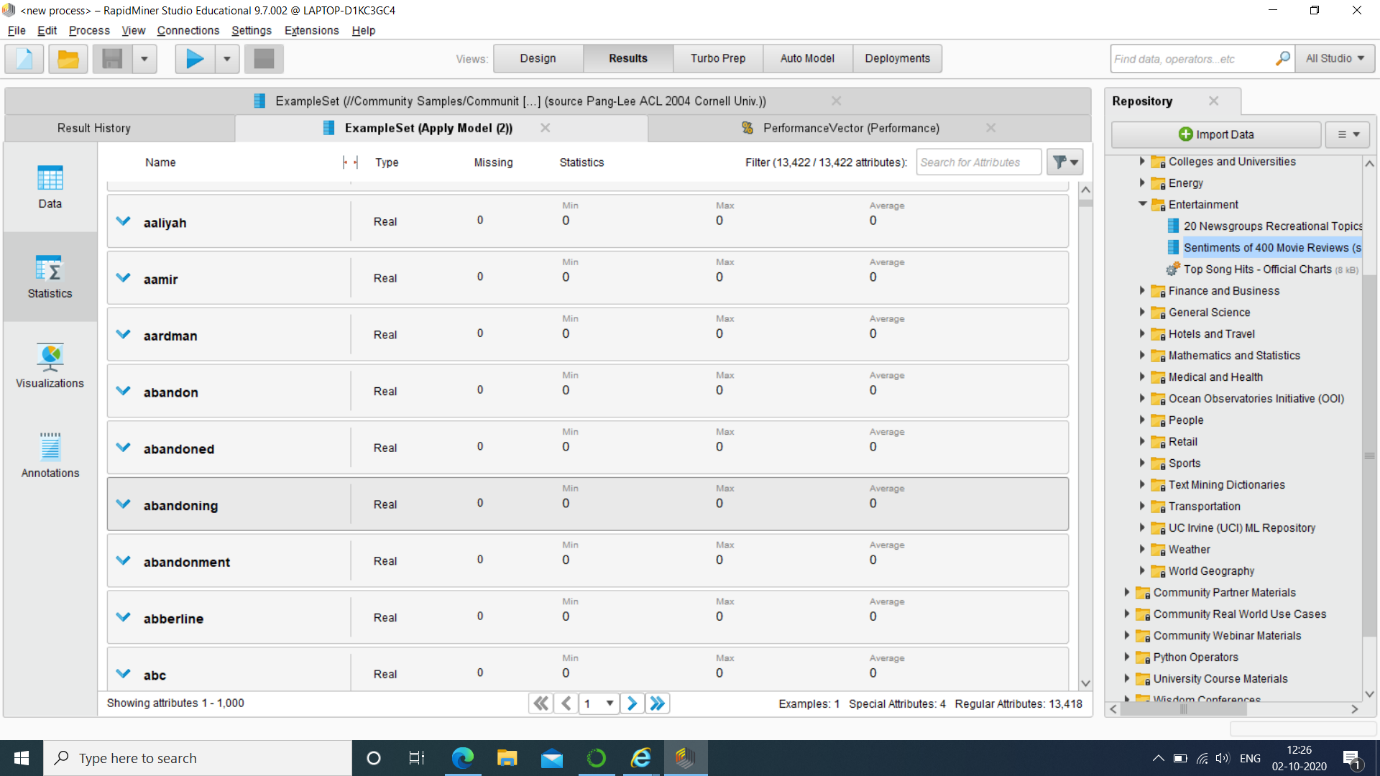
# Theory:

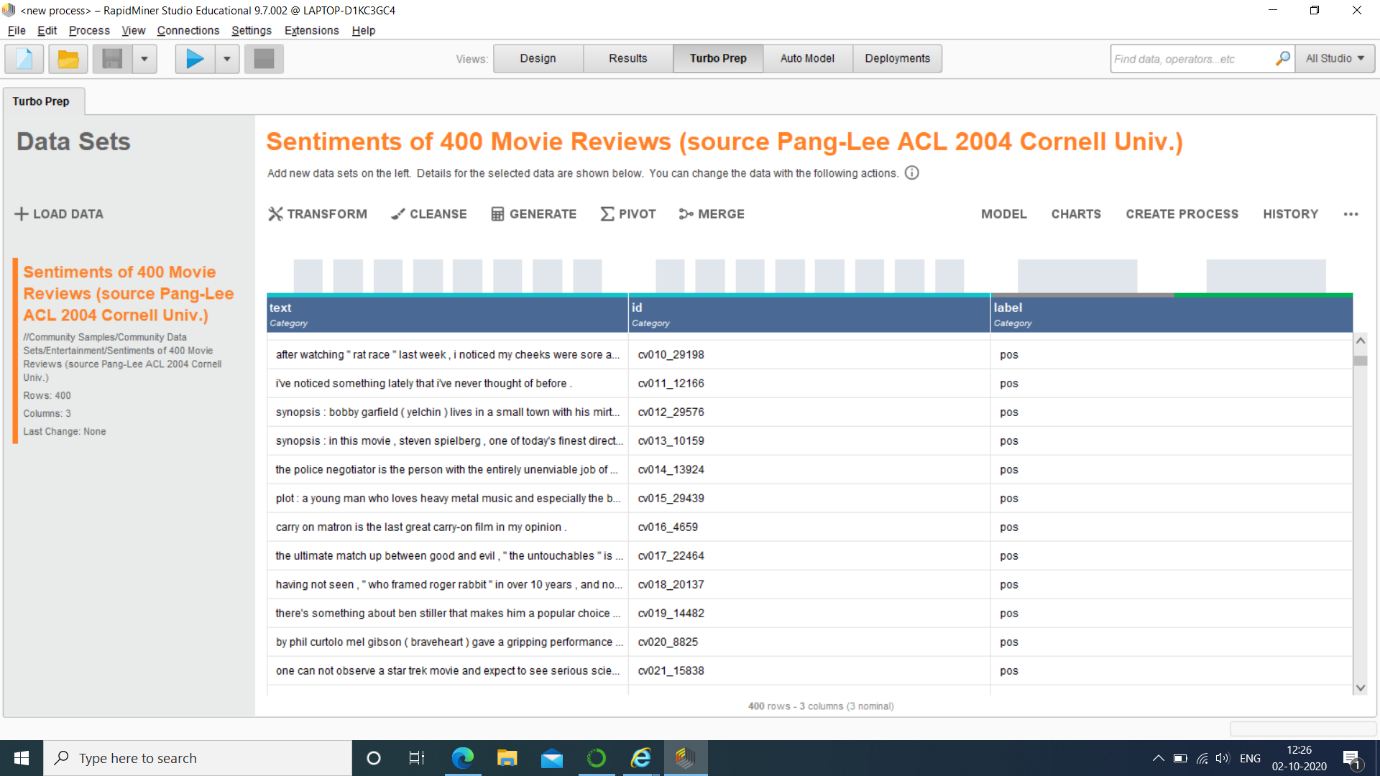
RapidMiner is a data science software platform that provides an integrated environment for data preparation, machine learning, deep learning, text mining, and predictive analytics. It is used for business and commercial applications as well as for research, education, training, rapid prototyping, and application development and supports all steps of the machine learning process including data preparation, results visualization, model validation and optimization. RapidMiner is developed on an open core model.

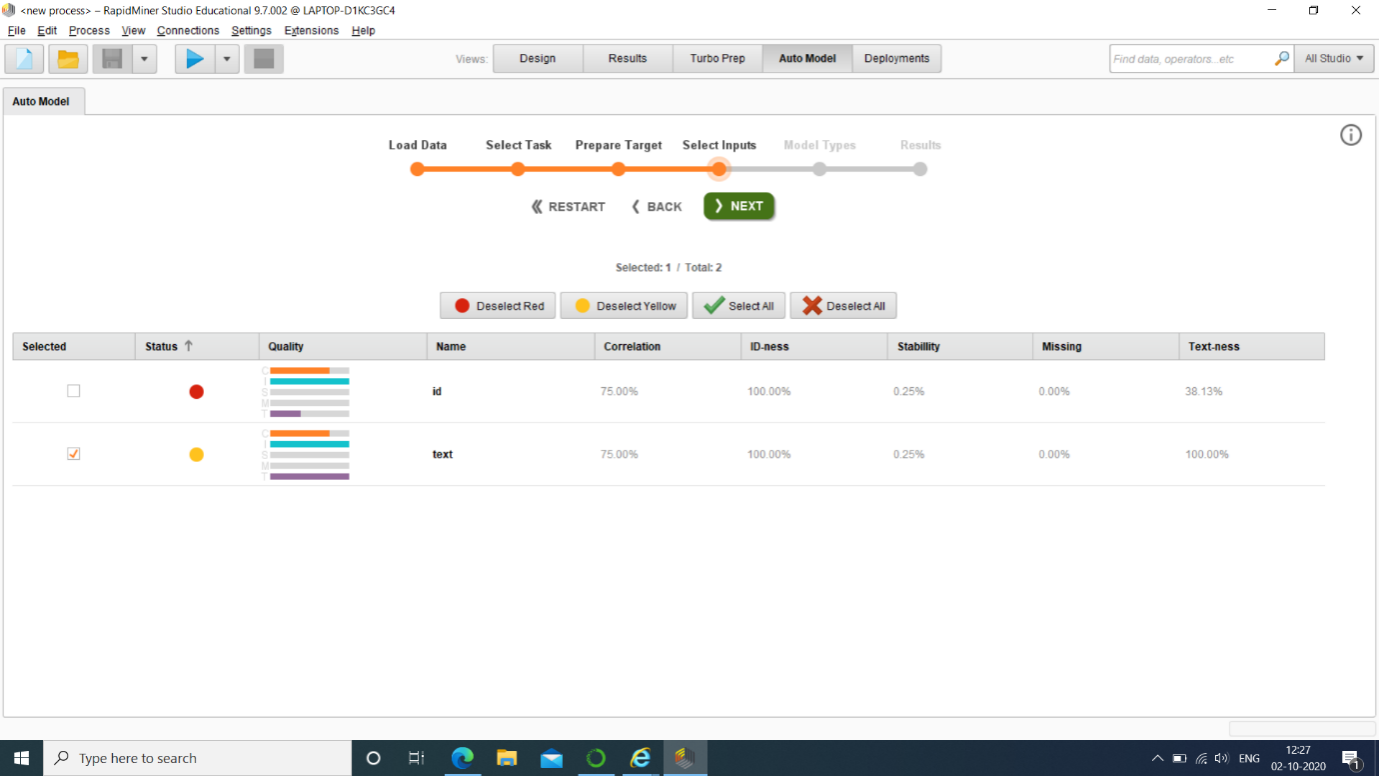












# Conclusion:

1. Sentiment analysis is often used in business to detect sentiment in

social data, gauge brand reputation, and understand customers.

2. Successfully build a model using Rapid Miner to analyse sentiments of

movies reviews.

3. Rapid Miner provides a great and easy interface for doing sentiment

analysis as well as for different ML modelling.