

# **CHAPTER-1**

## **INTRODUCTION**

### **1.1. SENTIMENT ANALYSIS**

Sentiment analysis (sometimes known as opinion mining or emotion AI) refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine.

Generally speaking, sentiment analysis aims to determine the attitude of a speaker, writer, or other subject with respect to some topic or the overall contextual polarity or emotional reaction to a document, interaction, or event. The attitude may be a judgment or evaluation (see appraisal theory), affective state (that is to say, the emotional state of the author or speaker), or the intended emotional communication (that is to say, the emotional effect intended by the author or interlocutor).

A basic task in sentiment analysis is classifying the polarity of a given text at the document, sentence, or feature/aspect level—whether the expressed opinion in a document, a sentence or an entity feature/aspect is positive, negative, or neutral. Advanced, "beyond polarity" sentiment classification looks, for instance, at emotional states such as "angry", "sad", and "happy".

### **1.2. TYPES**

#### **1.2.1. SUBJECTIVITY/OBJECTIVITY IDENTIFICATION**

This task is commonly defined as classifying a given text (usually a sentence) into one of two classes: objective or subjective. This problem can sometimes be more difficult than polarity classification. The subjectivity of words and phrases may depend on their context and an objective document may contain subjective sentences (e.g., a news article quoting people's opinions). Moreover, as mentioned by Su, results are largely dependent on the definition of subjectivity used when annotating texts. However, Pang showed that

removing objective sentences from a document before classifying its polarity helped improve performance.

### 1.2.2. FEATURE /ASPECT BASED

It refers to determining the opinions or sentiments expressed on different features or aspects of entities, e.g., of a cell phone, a digital camera, or a bank. A feature or aspect is an attribute or component of an entity, e.g., the screen of a cell phone, the service for a restaurant, or the picture quality of a camera. The advantage of feature-based sentiment analysis is the possibility to capture nuances about objects of interest. Different features can generate different sentiment responses, for example a hotel can have a convenient location, but mediocre food. This problem involves several sub-problems, e.g., identifying relevant entities, extracting their features/aspects, and determining whether an opinion expressed on each feature/aspect is positive, negative or neutral. The automatic identification of features can be performed with syntactic methods, with topic modeling, or with deep learning. More detailed discussions about this level of sentiment analysis can be found in Liu's work.

### 1.3. APPROACHES AND FEATURES

<b>SENTIMENT CLASSIFICATION APPROACHES</b>	<b>FEATURES/ TECHNIQUES</b>	<b>ADVANTAGES AND LIMITATIONS.</b>
MACHINE LEARNING	1.TERM PRESENCE AND FREQUENCY 2.PART OF SPEECH INFORMATION 3.NEGATIONS 4.OPIONION WORDS AND PHRASES	<b>ADVANTAGES:</b> The ability to adapt and create trained models for specific purposes and context.  <b>LIMITATIONS:</b> The low applicability to new data because it is necessary the availability of labeled data that could be costly or even prohibitive.
LEXICON BASED	1.MANUAL CONSTRUCTION 2.CORPUS BASED 3.DICTIONARY BASED	<b>ADVANTAGES:</b> Wider term coverage  <b>LIMITATIONS:</b> Finite number of words in lexicons and assignation of a fixed sentiment

		orientation and score of the word.
HYBRID BASED(MACHINE LEARNING+LEXICON BASED)	1.SENTIMENT LEXICON CONSTRUCTED USING PUBLIC RESOURCES FOR INTIAL SENTIMENT DETECTION . 2.SENTIMENTS WORDS AS FEATURES IN MACHINE LEARNING METHOD.	<b>ADVANTAGES:</b> Lexicon, detection and measurement of sentiments at the concept level and lesser sensitivity to change in topic domain  <b>LIMITATIONS</b> :Noisy review

Table 1: Comparison of different SA approaches

## 1.4. EVALUATION

The accuracy of a sentiment analysis system is, in principle, how well it agrees with human judgments. This is usually measured by variant measures based on precision and recall over the two target categories of negative and positive texts. However, according to research human raters typically only agree about 80% of the time. Thus, a program which achieves 70% accuracy in classifying sentiment is doing nearly as well as humans, even though such accuracy may not sound impressive. If a program were "right" 100% of the time, humans would still disagree with it about 20% of the time, since they disagree that much about any answer. On the other hand, computer systems will make very different errors than human assessors, and thus the figures are not entirely comparable. For instance, a computer system will have trouble with negations, exaggerations, jokes, and sarcasm, which are typically easy to handle for a human reader: some errors a computer system makes will seem overly naive to a human. In general, the utility for practical commercial tasks of sentiment analysis as it is defined in academic research has been called into question, mostly since the simple one-dimensional model of sentiment from negative to positive yields rather little actionable information for a client worrying about the effect of public discourse on e.g. brand or corporate reputation.

In recent years, to better fit market needs, evaluation of sentiment analysis has moved to more task-based measures, formulated together with representatives from PR agencies and market research professionals. The focus in e.g. the RepLab evaluation data set is less on the content of the text under consideration and more on the effect of the text in question on brand reputation.

## CHAPTER-2

### RELATED WORK

#### 2.1. PROBLEM STATEMENT

To illustrate a case study for examining the relation between negative sentiment and positive sentiment of twitter posts related to Prime Minister Narendra Modi. Over the past few days, most Indian Twitter users would have found their timelines abuzz with discussion around the people Prime Minister Narendra Modi followed on Twitter and their abusive tweets on change after Elections.

#### 2.2. PHASES

##### 2.2.1. PHASE 1

To apply the **pre existing algorithm** to the twitter data set and perform the sentimental analysis and gain results for the same.

Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine. Using sentiment analysis we can determine the emotional tone behind a series of words, use to gain an understanding of the attitudes, opinions and emotions expressed within an online mention for a particular happening or person.

Sentiment analysis refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information or the process of computationally identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writer's attitude towards a particular topic, product, etc. is positive, negative, or neutral.

### 2.2.2. PHASE 2

**Emoticons Detection:** Emojis are Unicode graphic symbols, used as a short-hand to express concepts and ideas. In contrast to the small number of well-known emoticons that carry clear emotional contents, there are hundreds of emojis.

Emoticons Detection and Recognition from text is a recent field of research that is closely related to Sentiment Analysis. Sentiment Analysis aims to detect positive, neutral, or negative feelings from text, whereas Emotion Analysis aims to detect and recognize types of feelings through the expression of texts, such as anger, disgust, fear, happiness, sadness, and surprise.

There are 6 emotion categories that are widely used to describe humans' basic emotions, based on facial expression: *anger*, *disgust*, *fear*, *happiness*, *sadness* and *surprise*. These are mainly associated with negative sentiment, with "Surprise" being the most ambiguous, as it can be associated with either positive or negative feelings. Interestingly, the number of basic human emotions has been recently "reduced", or rather re-categorized, to just 4; *happiness*, *sadness*, *fear/surprise*, and *anger/disgust*. It is surprising to many that we only have 4 basic emotions. For the sake of simplicity for this code story, we will use the more widely-used 6 emotions.

### 2.3. APPROACH USED

Lexicon-based sentiment analysis approach is used.

Application of a lexicon is one of the two main approaches to sentiment analysis and it involves calculating the sentiment from the semantic orientation of word or phrases that occur in a text. With this approach a dictionary of positive and negative words is required, with a positive or negative sentiment value assigned to each of the words. Different approaches to creating dictionaries have been proposed, including manual and automatic approaches. Generally speaking, in lexicon-based approaches a piece of text message is represented as a bag of words. Following this representation of the message, sentiment values from the dictionary are assigned to all positive and negative words or phrases within the message. A combining function, such as sum or average, is applied in order to make the final prediction regarding the overall sentiment for the message. Apart from a sentiment value, the aspect of the local context of a word is usually taken into consideration, such as negation or intensification.

In our work we have decided to apply a lexicon-based approach in order to avoid the need to generate a labeled training set. The main disadvantage of machine learning models is their reliance on labeled data. It is extremely difficult to ensure that sufficient and correctly labeled data can be obtained. Besides this, the fact that a lexicon-based approach can be more easily understood and modified by a human is considered a significant advantage for our work. We found it easier to generate an appropriate lexicon than collect and label relevant corpus. Given that the data pulled from social media are created by users from all over the globe, there is a limitation if the algorithm can only

handle English language. Consequently, sentiment analysis algorithm should be more easily transformable into different languages. Later in the paper we discuss how a lexicon-based sentiment analysis algorithm can be adapted to different languages by an appropriate translation of the sentiment lexicon and application of string similarity functions.

## **2.3.1. TWO MAIN PARTS OF LEXICON APPROACH**

### **2.3.1.1. LEXICON**

A lexicon, word-hoard or word-stock is the vocabulary of a person, language, or branch of knowledge. In linguistics, a lexicon is a language's inventory of lexemes.

Linguistic theories generally regard human languages as consisting of two parts: a lexicon, essentially a catalogue of a language's words (its word-stock); and a grammar, a system of rules which allow for the combination of those words into meaningful sentences. The lexicon is also thought to include bound morphemes, which cannot stand alone as words (such as most affixes). In some analyses, compound words and certain classes of idiomatic expressions and other collocations are also considered to be part of the lexicon. Dictionaries represent attempts at listing, in alphabetical order, the lexicon of a given language; usually, however, bound morphemes are not included.

#### **2.3.1.1.1. SIZE AND ORGANISATION**

Items in the lexicon are called lexemes, or lexical items, or word forms. Lexemes are not atomic elements but contain both phonological and morphological components. When describing the lexicon, a reductionist approach is used, trying to remain general while using a minimal description. To describe the size of a lexicon, lexemes are grouped into lemmas. A lemma is a group of lexemes generated by inflectional morphology. Lemmas are represented in dictionaries by headwords which list the citation forms and any irregular forms, since these must be learned to use the words correctly. Lexemes derived from a word by derivational morphology are considered new lemmas. The lexicon is also organized according to open and closed categories. Closed categories, such as determiners or pronouns, are rarely given new lexemes; their function is primarily syntactic. Open categories, such as nouns and verbs, have highly active generation mechanisms and their lexemes are more semantic in nature.

### **2.2.1.2. DATA DICTIONARY**

A crucial data set for any kind of text mining is a dictionary. As for sentiment analysis there are two big families of analysis algorithm. Both leverage dictionaries. The lexicon-based approach is where the ultimate score is calculated based on a per-word score from the dictionary and machine learning approach, where dictionaries are used to reduce data

dimensionality. A good dictionary for our purpose contains words bearing a strong positive or negative connotation, but does not contain neutral words. It may include two- or three-word phrases to improve handling negations and other common cases, like “can’t stand.” Ideally, the dictionary should come from the same topic and writing style that will be analyzed. (A lexicon for short texts like tweets written by teenagers will be quite different from a lexicon for diplomatic messages.)

**OR**

In other words:

1. A crucial data set for any kind of text mining is a dictionary.
2. The lexicon-based approach is where the ultimate score is calculated based on a per-word score from the dictionary.
3. A good dictionary for our purpose contains words bearing a strong positive or negative connotation, but does not contain neutral words.

## **2.4. NEED OF THE PROJECT**

Sentiment analysis is extremely useful in social media monitoring as it allows us to gain an overview of the wider public opinion behind certain topics. The applications of sentiment analysis are broad and powerful. The ability to extract insights from social data is a practice that is being widely adopted by organizations across the world.

**2.4.1. To Adjust Marketing Strategies:** the managerial perspective, social media is not just a platform where you can post and promote your services. It is a place where your customers’ chit-chat about your brand and is full of information about how brand is being perceived by your target customers. The information you get from Sentiment Analysis provides you with means to optimize your marketing strategy. By listening to what your customers feel and think about your brand, you can adjust your high-level messaging to meet their needs.

**2.4.2. To Measure ROI of Marketing Companion:** your audience. By combining the quantitative and the qualitative measurements, you can Success of your marketing campaign is not measured only by the increase in the number of followers, likes, or comments. The success also lies in how much positive discussion you are able to help facilitate amongst your customers. By doing sentiment analysis you can to see how much positive or negative discussions have occurred amongst measure the true ROI of your marketing campaign.

**2.4.3. Develop Product Quality :** Sentimental Analysis helps you complete your market research by getting to know what your customers' opinions are about your products/services and how you can align your products/services’ quality and features with their tastes.

**2.4.4. Improve Customer Service:** There are many factors that contribute to great customer service, such as on-time delivery, being responsive in social media, and adequate compensation for product's errors. Sentiment analysis can pick up negative discussions, and give you real-time alerts so that you can respond quickly.

**2.3.5.** Using sentiment analysis we can **determine the emotional tone** behind a series of words, use to gain an understanding of the attitudes, opinions and emotions expressed within an online mention for a particular happening or person



## **CHAPTER-3**

### **SYSTEM ANALYSIS AND DESIGN**

#### **3.1. SYSTEM REQUIREMENT SPECIFICATION**

##### **3.1.1. SOFTWARE REQUIREMENTS**

- Operating System : Windows XP/2003 or Higher Version of Windows OS, Linux
- User Interface : Python command line.
- Programming Language : Python, SQL
- Python 3.x
- IDE/Workbench : PyCharm 2017, Python Editor IDE, DB Browser
- Database : SQLite

##### **3.1.2. HARDWARE REQUIREMENTS**

- Processor : Pentium IV or above.
- Hard Disk : 50GB
- RAM : 512 MB or more

### 3.2. E-R DIAGRAMS

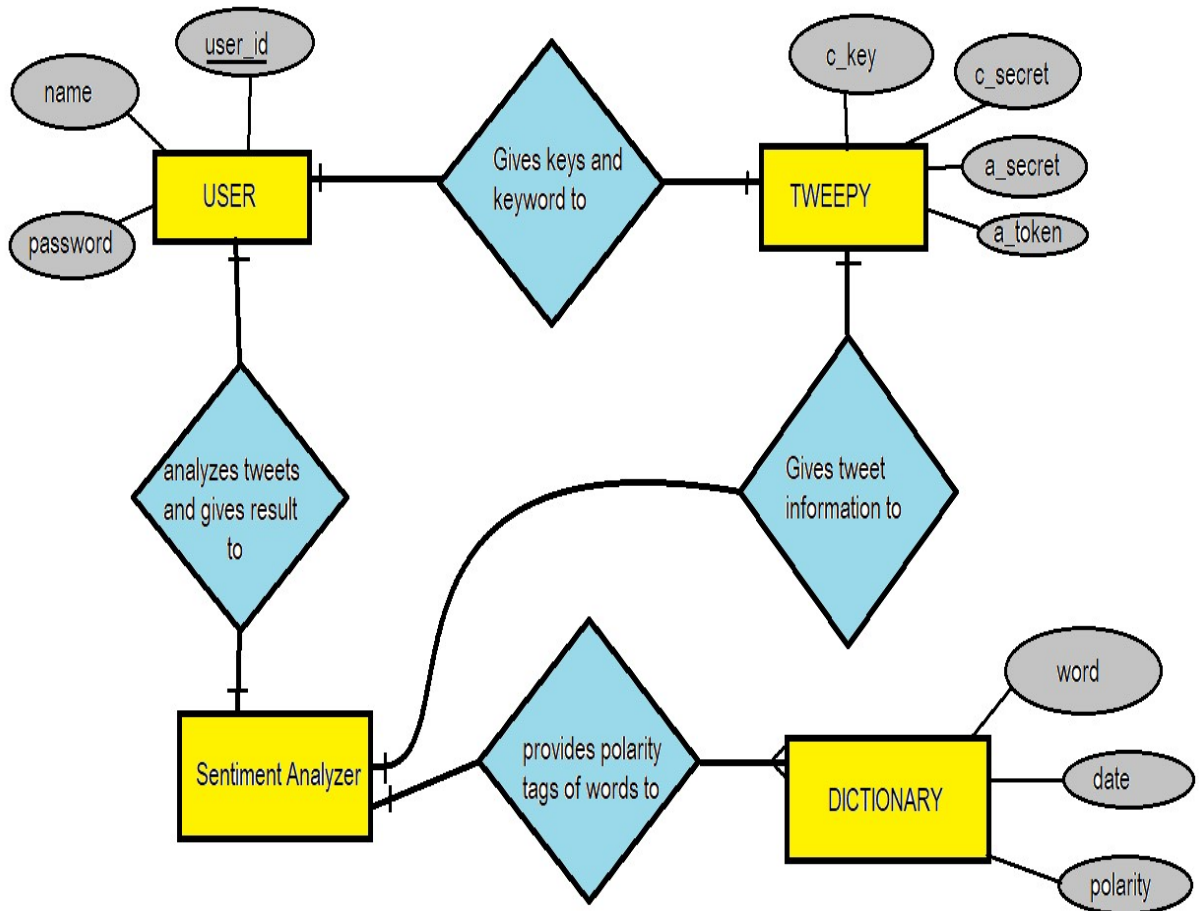


Figure 1: ER Diagram

### 3.3. 0 - LEVEL DFD

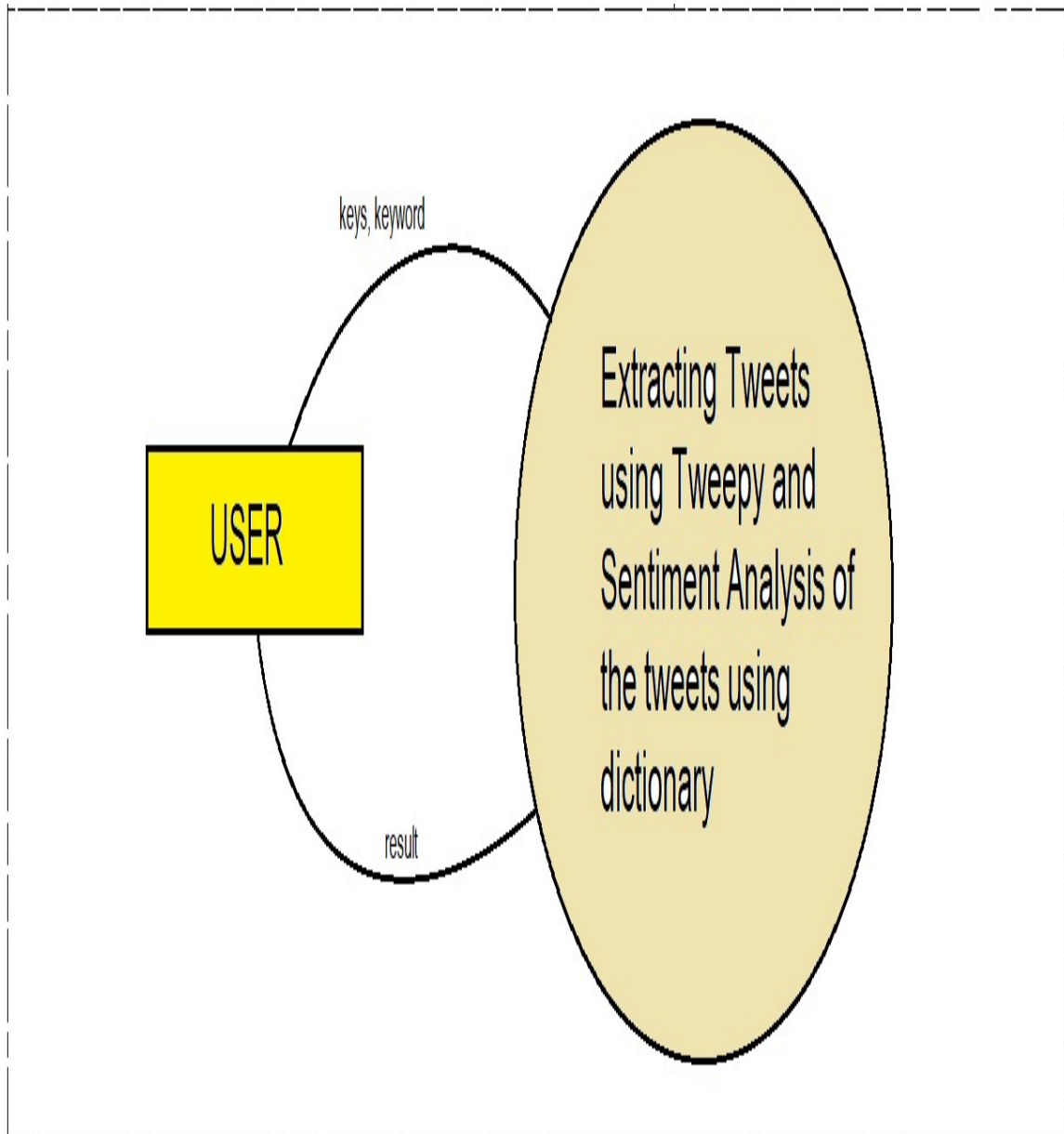


Figure 2: Level-0 DFD

### 3.4. 1-LEVEL DFD (PHASE 1)

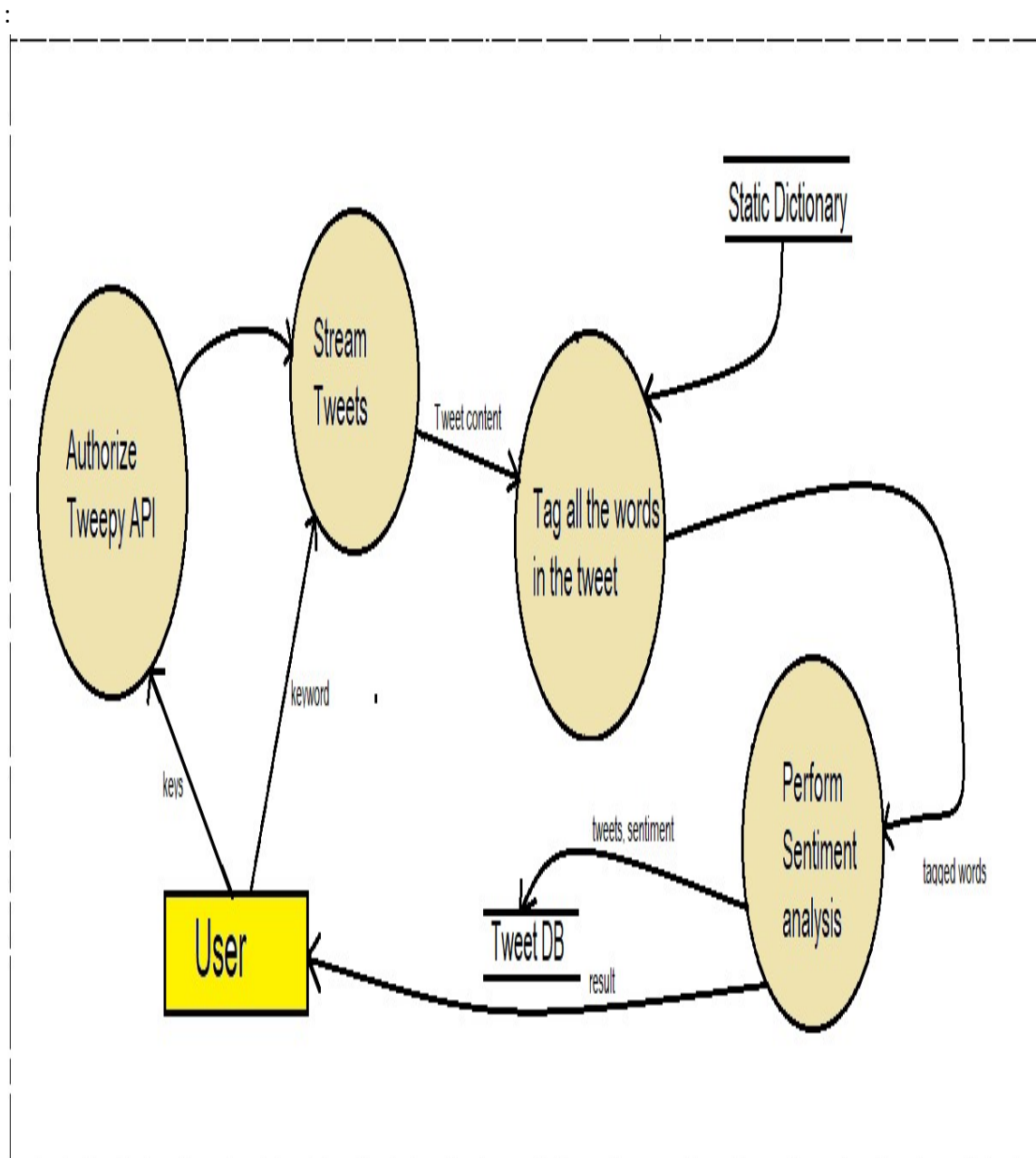


Figure 3: Phase I, Level-1 DFD

### 3.4. 1-LEVEL DFD(PHASE 2)

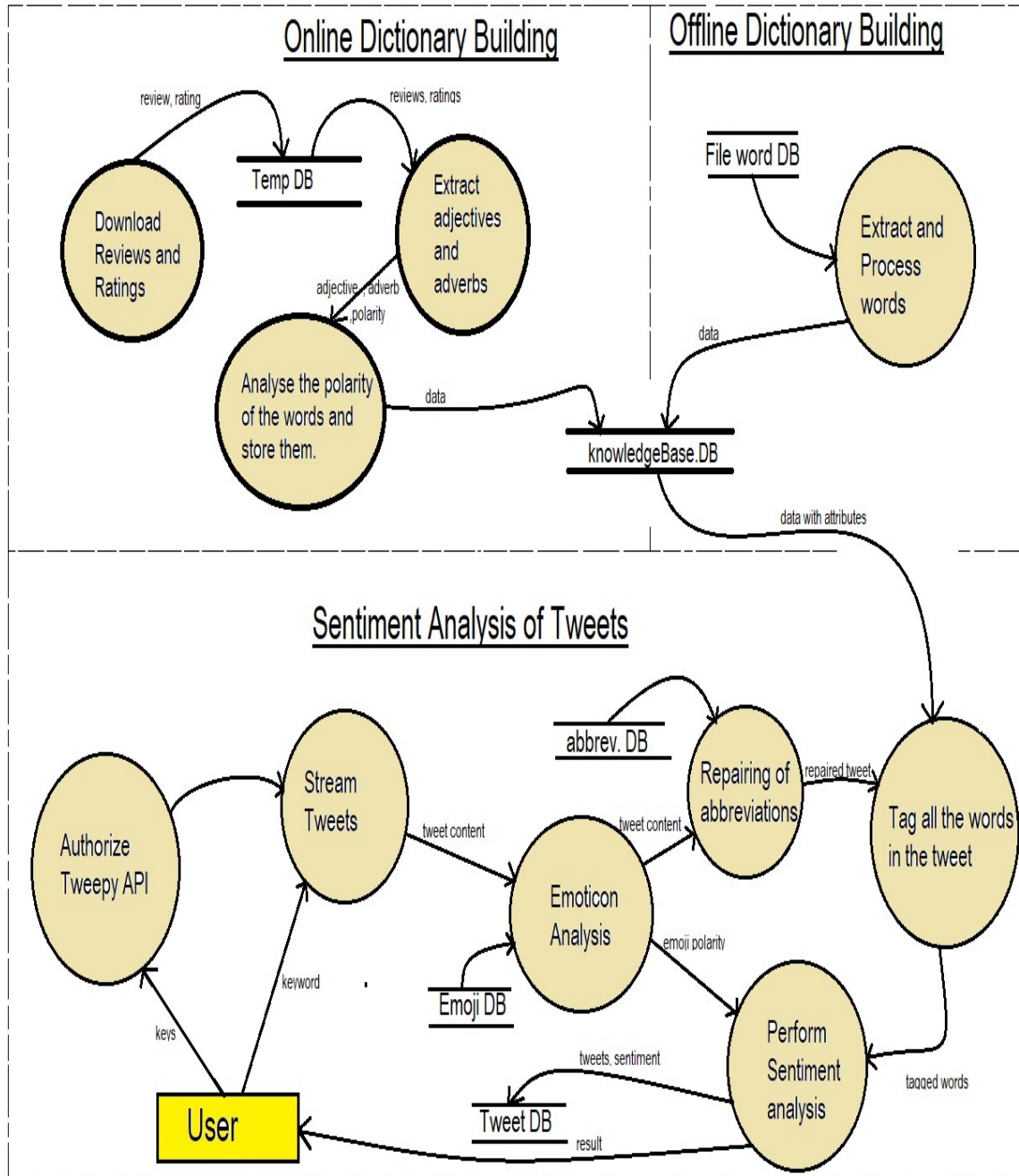


Figure 4: Phase II, Level-1 DFD

## **CHAPTER-4**

### **PROPOSED WORK**

#### **4.1. PROPOSED ALGORITHM**

1. Collect reviews of items or movies with ratings from the internet.
2. If the rating of the content is above 6, rate it as positive but if it's below 5, rate it negative.
3. Build word database and multiplier database using the content:
  - a. Find all adjectives in the content.
  - b. For each adjective, if it already exists: add +1 for positive and subtract -1 for negative.
  - c. If it doesn't exist, add it to the word database with the given polarity.
  - d. Find all adverbs in the content.
  - e. For each adverb, if it already exists: multiply with 2 for positive and divide by 2 for negative.
  - f. If it doesn't exist, add it to the multiplier database with the given polarity.
4. Stream tweets using Twitter API and for each tweet follow steps 5 to 12.
5. Repair words like hlo (hello), wat (what) etc.

6. Find all negative and positive emojis and rate them accordingly.
7. Split the tweet into sentences and words and tag them.
8. Build the sentiment dictionary from the database containing positive, negative, incremental, decremental and inversive words.
9. Tag all the words according to the sentiment dictionary
10. Analyze the tagged words and find the sum of all the sentiments in the sentences.
11. Add emoji sentiment ratings to the previous result.
12. Display result.

Note: Step 1 to 3 has to be stored as a separate program and to be run every week to update the dictionary.

## 4.2. PSUEDOCODE (PHASE 1)

### **Sentiment\_analyzer.py:**

```
import nltk, yaml
class Splitter(object):
    def __init__(self)
    def split(self, text)

class POSTagger(object):
    def __init__(self)
    def pos_tag(self, sentences)

class DictionaryTagger(object):
    def __init__(self, dictionary_paths)
    def tag(self, postagged_sentences)
```

```

def tag_sentence(self, sentence, tag_with_lemmas=False)

def value_of(sentiment)

def sentence_score(sentence_tokens, previous_token, acum_score)

def sentiment_score(review)

```

---

## Twitter.py:

```

from tweepy import Stream, OAuthHandler
from tweepy.streaming import StreamListener
import time, urllib, re
from textblob import TextBlob
from sentiment_analyzer import Splitter, POSTagger, DictionaryTagger,
value_of
from sentiment_analyzer import sentence_score, sentiment_score

ckey = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'
csecret = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'
atoken = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx'
asecret = 'yyyyyyyyyyyyyyyyyyyy'
def get_sentiment(tweet)

class listener(StreamListener):
    def __init__(self):
    def on_data(self, data):
        quit_if_limit_reached()
        text = extract_tweet()
        splitter = Splitter()
        postagger = POSTagger()
        splitted_sentences = splitter.split(text)
        pos_tagged_sentences = postagger.pos_tag(splitted_sentences)
        dicttagger = DictionaryTagger(dictionary_paths)
        dict_tagged_sentences = dicttagger.tag(pos_tagged_sentences)
        senti = sentiment_score(dict_tagged_sentences)
        analyse_other_tweet_data()
        try:
            store_and_print_result()
        except BaseException:
            print('Failed')
            time.sleep(5)
            return False
        return True
    def on_error(self, status):
        print(status)

auth = OAuthHandler(ckey, csecret)
auth.set_access_token(atoken, asecret)
l = listener()
twitterStream = Stream(auth, l)
twitterStream.filter(track=['Modi'])

```



### 4.3. PSUEDOCODE (PHASE 2)

#### KnowledgeBase.py:

```
import nltk
import re
import time
try:
    import urllib.request as urllib2
except ImportError:
    import urllib2
try:
    from urllib.request import urlopen
except ImportError:
    from urllib2 import urlopen
import http.cookiejar as cookielib
from http.cookiejar import CookieJar
import datetime
import sqlite3
import yaml

cj = CookieJar()
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
opener.addheaders = [('User-agent', 'Chrome/61.0')]
conn = sqlite3.connect('knowledgeBase.db')
c = conn.cursor()
site = 'http://www.imdb.com'
contentArray = [ ]
ratingArray = [ ]
def loadDicts(dictionary_paths)

def IMDBcontent()

def buildWordBase(dictionary)

def buildMultiplierBase(dictionary)

dict_paths      =      [      'dicts/pos_dict.yml',      'dicts/newpos.yml',
'dicts/pos.yml', 'dicts/pos2.yml',
                        'dicts/neg_dict.yml', 'dicts/newneg.yml']
multiplier_dict = ['dicts/inc.yml', 'dicts/dec.yml', 'dicts/inv.yml']
#dictionary = loadDicts(multiplier_dict)
#buildMultiplierBase(dictionary)
IMDBcontent()
site = 'https://www.amazon.in'
aSearch      =      'https://www.amazon.in/s/ref=nb_sb_noss_2?url=search-
alias%3Daps&field-keywords='
keyword = 'phones'
def amazonContent()

#amazonContent()

def buildBase(polarity, content)
```

```
def processContent():
```

```
processContent()
```

---

### **Features.py:**

```
import re
import json
```

```
class Emoticons:
    def analyse(self, string)
```

```
def repairString(string)
```

---

### **sentiment\_analyzer.py:**

```
import nltk,yaml
from emo import split_emo
import datetime
import sqlite3
import features
```

```
class Splitter(object):
    def __init__(self)
    def split(self, text)
```

```
class POSTagger(object):
    def __init__(self)
    def pos_tag(self, sentences)
```

```
class DictionaryTagger(object):
    def __init__(self)
    def tag(self, postagged_sentences)
    def tag_sentence(self, sentence, tag_with_lemmas=False)
```

```
def value_of(sentiment)
```

```
def sentence_score(sentence_tokens, previous_token, acum_score)
```

```
def sentiment_score(review)
```

---

### **Twitter.py:**

```
from tweepy import Stream, OAuthHandler
```

```

from tweepy.streaming import StreamListener
import time, urllib, re
from textblob import TextBlob
from sentiment_analyser import Splitter, POSTagger, DictionaryTagger,
value_of
from sentiment_analyser import sentence_score, sentiment_score
import features

ckey = 'xxxxxxxxxxxxxxxxkxxxxxxxxxxxxx'
csecret = 'cccccccccccccccccaaaaaaaaaaasssssssssssssss'
atoken = 'aaaaaaaaaaaaaa-ttttttttttttttttttttttttttttttttttttt'
asecret = 'aaaaaaaaaaaaaaaaaasssssssssssssssssssssssssssssssssxx'

def get_sentiment(tweet)

class listener(StreamListener):
    def __init__(self):
    def on_data(self, data):
        print_final_result_and_quit_if_limit_reached()
        text = str(tweet)
        splitter = Splitter()
        postagger = POSTagger()
        emoji = features.Emoticons()
        emos = emoji.analyse(text)
        text = features.repairString(text)
        splitted_sentences = splitter.split(text)
        pos_tagged_sentences = postagger.pos_tag(splitted_sentences)
        dicttagger = DictionaryTagger()
        dict_tagged_sentences = dicttagger.tag(pos_tagged_sentences)
        senti = sentiment_score(dict_tagged_sentences) +
        emos['positive'] - emos['negative']
        analyse_other_data()
        try:
            storing_and_printing_result()
        except BaseException:
            print('Failed')
            time.sleep(5)
            return False
        return True

    def on_error(self, status):
        print(status)

auth = OAuthHandler(ckey, csecret)
auth.set_access_token(atoken, asecret)
l = listener()
twitterStream = Stream(auth, l)
twitterStream.filter(track=['Modi'])

```

## CHAPTER-5

### IMPLEMENTATION/RESULT

#### 5.1. PHASE 1

##### 5.1.1. CODE

###### Sentiment\_analyzer.py:

```
import nltk,yaml

#emodict = {':': ['positive'], '\': ['negative'], '(': ['negative'], '*': ['positive'], 'D': ['positive']}

class Splitter(object):
    def __init__(self):
        self.nltk_splitter =
nltk.data.load('tokenizers/punkt/english.pickle')
        self.nltk_tokenizer = nltk.tokenize.TreebankWordTokenizer()

    def split(self, text):
        """
        input format: a paragraph of text
        output format: a list of lists of words.
        e.g.: [['this', 'is', 'a', 'sentence'], ['this', 'is',
'another', 'one']]
        """
        sentences = self.nltk_splitter.tokenize(text)
        tokenized_sentences = [self.nltk_tokenizer.tokenize(sent) for
sent in sentences]
        return tokenized_sentences

class POSTagger(object):
    def __init__(self):
        pass

    def pos_tag(self, sentences):
        """
        input format: list of lists of words
        e.g.: [['this', 'is', 'a', 'sentence'], ['this', 'is',
'another', 'one']]
        output format: list of lists of tagged tokens. Each tagged
tokens has a
```

```

        form, a lemma, and a list of tags
        e.g: [['this', 'this', ['DT']], ('is', 'be', ['VB']),
('a', 'a', ['DT']), ('sentence', 'sentence', ['NN'])],
            [['this', 'this', ['DT']], ('is', 'be', ['VB']),
('another', 'another', ['DT']), ('one', 'one', ['CARD'])]]
    """

```

```

    pos = [nltk.pos_tag(sentence) for sentence in sentences]
    #adapt format
    pos = [(word, word, [postag]) for (word, postag) in sentence]
for sentence in pos]
    return pos

```

```

class DictionaryTagger(object):
    def __init__(self, dictionary_paths):
        files = [open(path, 'r') for path in dictionary_paths]
        dictionaries = [yaml.load(dict_file) for dict_file in files]
        map(lambda x: x.close(), files)
        self.dictionary = {}
        self.max_key_size = 0
        for curr_dict in dictionaries:
            for key in curr_dict:
                if key in self.dictionary:
                    self.dictionary[key].extend(curr_dict[key])
                else:
                    self.dictionary[key] = curr_dict[key]
                    self.max_key_size = max(self.max_key_size,
len(key))

```

```

    def tag(self, postagged_sentences):
        return [self.tag_sentence(sentence) for sentence in
postagged_sentences]

```

```

    def tag_sentence(self, sentence, tag_with_lemmas=False):
        """
        the result is only one tagging of all the possible ones.
        The resulting tagging is determined by these two priority
rules:
            - longest matches have higher priority
            - search is made from left to right
        """
        tag_sentence = []
        N = len(sentence)
        if self.max_key_size == 0:
            self.max_key_size = N
        i = 0
        while (i < N):
            j = min(i + self.max_key_size, N) #avoid overflow
            tagged = False
            while (j > i):
                expression_form = ' '.join([word[0] for word in
sentence[i:j]]).lower()
                expression_lemma = ' '.join([word[1] for word in
sentence[i:j]]).lower()
                if tag_with_lemmas:

```

```

        literal = expression_lemma
    else:
        literal = expression_form
    if literal in self.dictionary:
        is_single_token = j - i == 1
        original_position = i
        i = j
        taggings = [tag for tag in
self.dictionary[literal]]
        tagged_expression = (expression_form,
expression_lemma, taggings)
        if is_single_token:
            original_token_tagging =
sentence[original_position][2]
tagged_expression[2].extend(original_token_tagging)
        tag_sentence.append(tagged_expression)
        tagged = True
    else:
        j = j - 1
    if not tagged:
        tag_sentence.append(sentence[i])
        i += 1
    return tag_sentence

```

```

def value_of(sentiment):
    if sentiment == 'positive': return 1
    if sentiment == 'negative': return -1
    return 0

```

```

def sentence_score(sentence_tokens, previous_token, acum_score):
    if not sentence_tokens:
        return acum_score
    else:
        current_token = sentence_tokens[0]
        tags = current_token[2]
        token_score = sum([value_of(tag) for tag in tags])
        if previous_token is not None:
            previous_tags = previous_token[2]
            if 'inc' in previous_tags:
                token_score *= 2.0
            elif 'dec' in previous_tags:
                token_score /= 2.0
            elif 'inv' in previous_tags:
                token_score *= -1.0
        return sentence_score(sentence_tokens[1:], current_token,
acum_score + token_score)

```

```

def sentiment_score(review):
    return sum([sentence_score(sentence, None, 0.0) for sentence in
review])

```

```

##text = """What can I say about this place. The staff of the
restaurant is nice and the eggplant is not bad. Apart from that, very

```

```

uninspired food, lack of atmosphere and too expensive. :( I am a
staunch vegetarian and was sorely dissapointed with the veggie options
on the menu. Will be the last time I visit, I recommend others to
avoid."""
##text = """This is bullshit!!"""
##splitter = Splitter()
##postagger = POSTagger()
##
##splitted_sentences = splitter.split(text)
##
##print(splitted_sentences)
##print('\n\n')
##pos_tagged_sentences = postagger.pos_tag(splitted_sentences)
##
##print(pos_tagged_sentences)
##
##
##dicttagger = DictionaryTagger(['dicts/pos_dict.yml',
'dicts/newpos.yml', 'dicts/pos.yml', 'dicts/pos2.yml',
'dicts/neg_dict.yml', 'dicts/newneg.yml', 'dicts/inc.yml',
'dicts/dec.yml', 'dicts/inv.yml'])
##
##dict_tagged_sentences = dicttagger.tag(pos_tagged_sentences)
##
##print(dict_tagged_sentences)
##
##print(sentiment_score(dict_tagged_sentences))

```

---

## Twitter.py:

```

from tweepy import Stream, OAuthHandler
from tweepy.streaming import StreamListener
import time, urllib, re
from textblob import TextBlob
from sentiment_analyzer import Splitter, POSTagger, DictionaryTagger,
value_of
from sentiment_analyzer import sentence_score, sentiment_score

ckey = 'McfEpmGoArTagly0YEgLmAw9B'
csecret = 'nhh8YrANBISlCcJm6TlXZr9dDuBOFBK69fyR3vIISlNKQUzmTF'
atoken = '580508392-TGvrelm0EqXppjxZoiNoB7B4bfDBxG6lm4rOwIdo'
asecret = 'ZWLk0U1KilShIZHeJnYGshMhxI1lJLHL9WurC9IsNck0m'

def get_sentiment(tweet):
    analysis = TextBlob(tweet)
    return analysis.sentiment.polarity

class listener(StreamListener):
    def __init__(self):
        self.count = 0
        self.positive_score = 0.0

```

```

self.p_count = 0.0
self.n_count = 0.0
self.neutral = 0.0
self.negative_score = 0.0

def on_data(self, data):
    self.count+=1
    if self.count>5:
        print('\n\n=====')
        print('ANALYSIS COMPLETE: ')
        print('=====')
        print('Average sentiment score on the Modi:
'+str((self.positive_score+self.negative_score)/5))
        print('Percentage of people who gave +ve tweets:
'+str((100*self.p_count/5)))
        print('Percentage of people who gave -ve tweets:
'+str((100*self.n_count/5)))
        print('Percentage of people who gave neutral tweets:
'+str((100*self.neutral/5)))
        exit(0)
    tweet = data.split(', "text":') [1].split(', "source":') [0]
    text = str(tweet)
    splitter = Splitter()
    postagger = POSTagger()
    splitted_sentences = splitter.split(text)
    pos_tagged_sentences = postagger.pos_tag(splitted_sentences)
    dicttagger = DictionaryTagger(['dicts/pos_dict.yml',
'dicts/newpos.yml', 'dicts/pos.yml', 'dicts/pos2.yml',
'dicts/neg_dict.yml', 'dicts/newneg.yml', 'dicts/inc.yml',
'dicts/dec.yml', 'dicts/inv.yml'])
    dict_tagged_sentences = dicttagger.tag(pos_tagged_sentences)
    senti = sentiment_score(dict_tagged_sentences)
    if senti>0:
        self.positive_score+=senti
        self.p_count+=1
    elif senti<0:
        self.negative_score+=senti
        self.n_count+=1
    else:
        self.neutral+=1
    date = data.split('{"created_at":') [1].split('"', "id":') [0]
    favorites =
int(data.split(', "favorite_count":') [1].split(', "entities":') [0])
    followers =
int(data.split(', "followers_count":') [1].split(', "friends_count":') [0])
    if followers!=0:
        interest_level = 100*favorites/followers
    else:
        interest_level = 0
    retweets =
data.split(', "retweet_count":') [1].split(', "favorite_count":') [0]
    try:
        file = open('tDB.csv', 'a')
        file.write('Feed: \n')
        file.write(data)
        file.write('\n')
        file.close()

```



```

        print(len(data))
        print('Tweet: ', text, '\nDate: ', date, '\nFavorites: ',
favorites, '\nRetweets: ', retweets, '\nSentiment Rating: ', senti,
'\nInterest Level: ', "%.2f"%interest_level,'%')
    except BaseException:
        print('Failed')
        time.sleep(5)
        return False
    return True

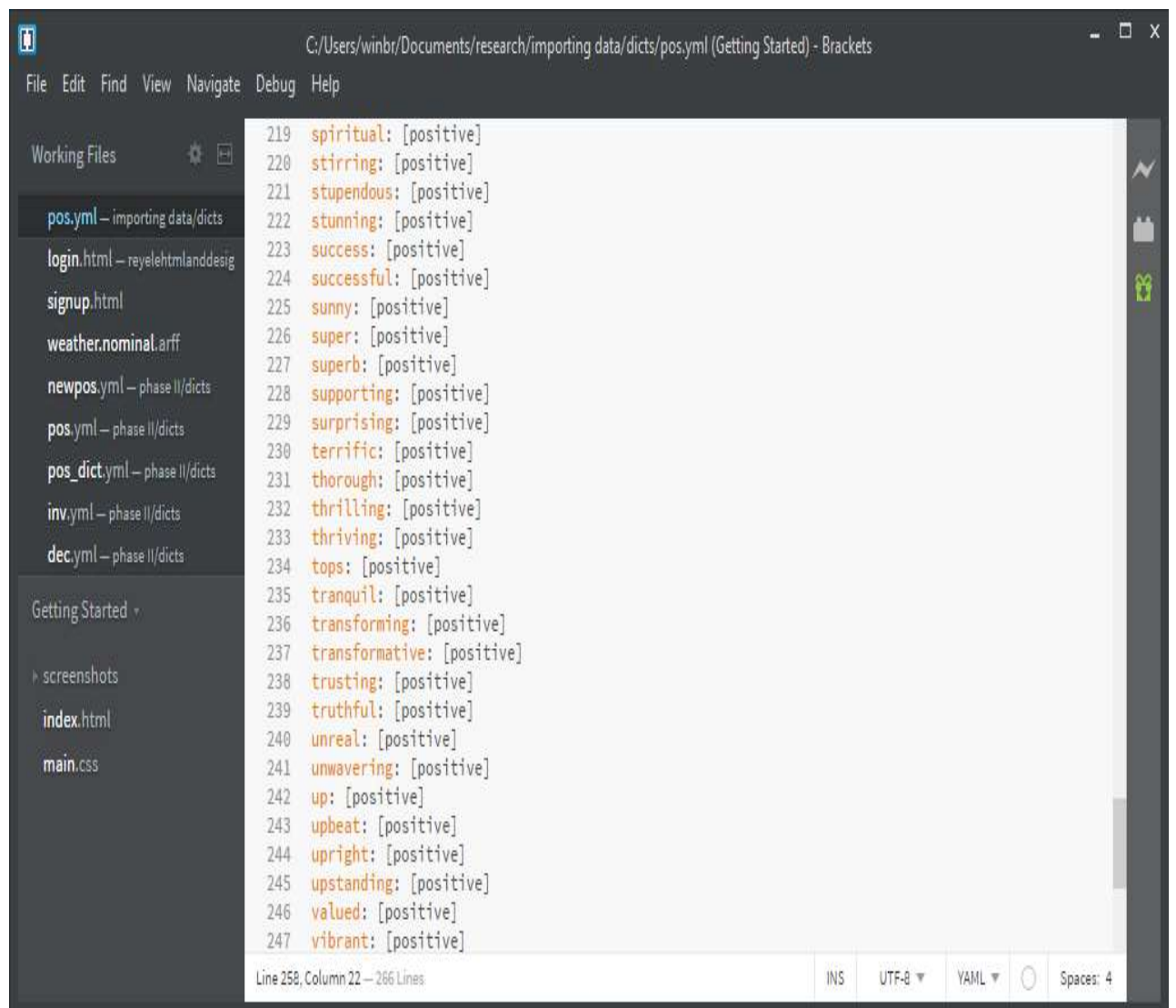
def on_error(self, status):
    print(status)

auth = OAuthHandler(ckey, csecret)
auth.set_access_token(accessToken, asecret)
l = listener()
twitterStream = Stream(auth, l)
twitterStream.filter(track=['Modi'])

```

## 5.1.2. OUTPUTS

### 5.1.2.1. DATA DICTIONARY



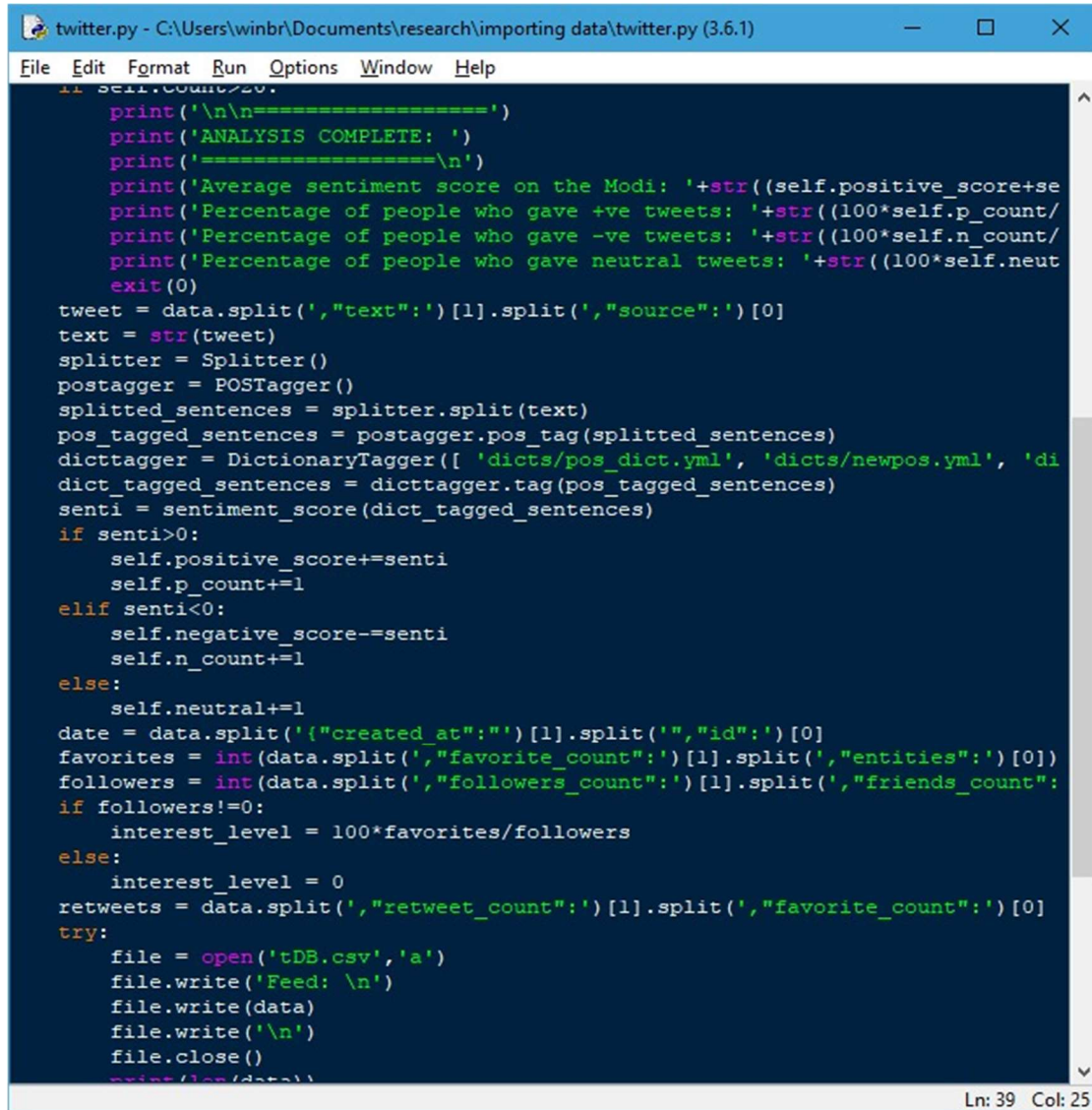
The screenshot shows the Brackets code editor interface. The title bar indicates the file path: `C:/Users/winbr/Documents/research/importing data/dicts/pos.yml (Getting Started) - Brackets`. The menu bar includes File, Edit, Find, View, Navigate, Debug, and Help. The left sidebar shows a 'Working Files' list with several files, including `pos.yml - importing data/dicts`, which is the active file. The main editor area displays the content of `pos.yml`, which is a list of words and their sentiment labels in a YAML format. The text is as follows:

```
219 spiritual: [positive]
220 stirring: [positive]
221 stupendous: [positive]
222 stunning: [positive]
223 success: [positive]
224 successful: [positive]
225 sunny: [positive]
226 super: [positive]
227 superb: [positive]
228 supporting: [positive]
229 surprising: [positive]
230 terrific: [positive]
231 thorough: [positive]
232 thrilling: [positive]
233 thriving: [positive]
234 tops: [positive]
235 tranquil: [positive]
236 transforming: [positive]
237 transformative: [positive]
238 trusting: [positive]
239 truthful: [positive]
240 unreal: [positive]
241 unwavering: [positive]
242 up: [positive]
243 upbeat: [positive]
244 upright: [positive]
245 upstanding: [positive]
246 valued: [positive]
247 vibrant: [positive]
```

The status bar at the bottom shows 'Line 258, Column 22 - 266 Lines', 'INS', 'UTF-8', 'YAML', and 'Spaces: 4'.

Figure 5: Data Dictionary

### 5.1.2.2. MAIN CODE



```
twitter.py - C:\Users\winbr\Documents\research\importing data\twitter.py (3.6.1)
File Edit Format Run Options Window Help
11 self.count+=1
    print('\n\n=====')
    print('ANALYSIS COMPLETE: ')
    print('=====\n')
    print('Average sentiment score on the Modi: '+str((self.positive_score+self.negative_score)/self.count))
    print('Percentage of people who gave +ve tweets: '+str((100*self.p_count/self.count)))
    print('Percentage of people who gave -ve tweets: '+str((100*self.n_count/self.count)))
    print('Percentage of people who gave neutral tweets: '+str((100*self.neut_count/self.count)))
    exit(0)

tweet = data.split(', "text":')[1].split(', "source":')[0]
text = str(tweet)
splitter = Splitter()
postagger = POSTagger()
splitted_sentences = splitter.split(text)
pos_tagged_sentences = postagger.pos_tag(splitted_sentences)
dicttagger = DictionaryTagger(['dicts/pos_dict.yml', 'dicts/newpos.yml', 'dicts/stop_dict.yml'])
dict_tagged_sentences = dicttagger.tag(pos_tagged_sentences)
senti = sentiment_score(dict_tagged_sentences)
if senti>0:
    self.positive_score+=senti
    self.p_count+=1
elif senti<0:
    self.negative_score-=senti
    self.n_count+=1
else:
    self.neutral+=1
date = data.split('{ "created_at":')[1].split(', "id":')[0]
favorites = int(data.split(', "favorite_count":')[1].split(', "entities":')[0])
followers = int(data.split(', "followers_count":')[1].split(', "friends_count":')[0])
if followers!=0:
    interest_level = 100*favorites/followers
else:
    interest_level = 0
retweets = data.split(', "retweet_count":')[1].split(', "favorite_count":')[0]
try:
    file = open('tDB.csv', 'a')
    file.write('Feed: \n')
    file.write(data)
    file.write('\n')
    file.close()
except:
    print('Error: ')
print('Log: ')
print(data)
```

Ln: 39 Col: 25

Figure 6: Main Code

```
sentiment_analyzer.py - C:\Users\winbr\Documents\research\importing data\sentiment_analyzer.py (3.6.1)
File Edit Format Run Options Window Help

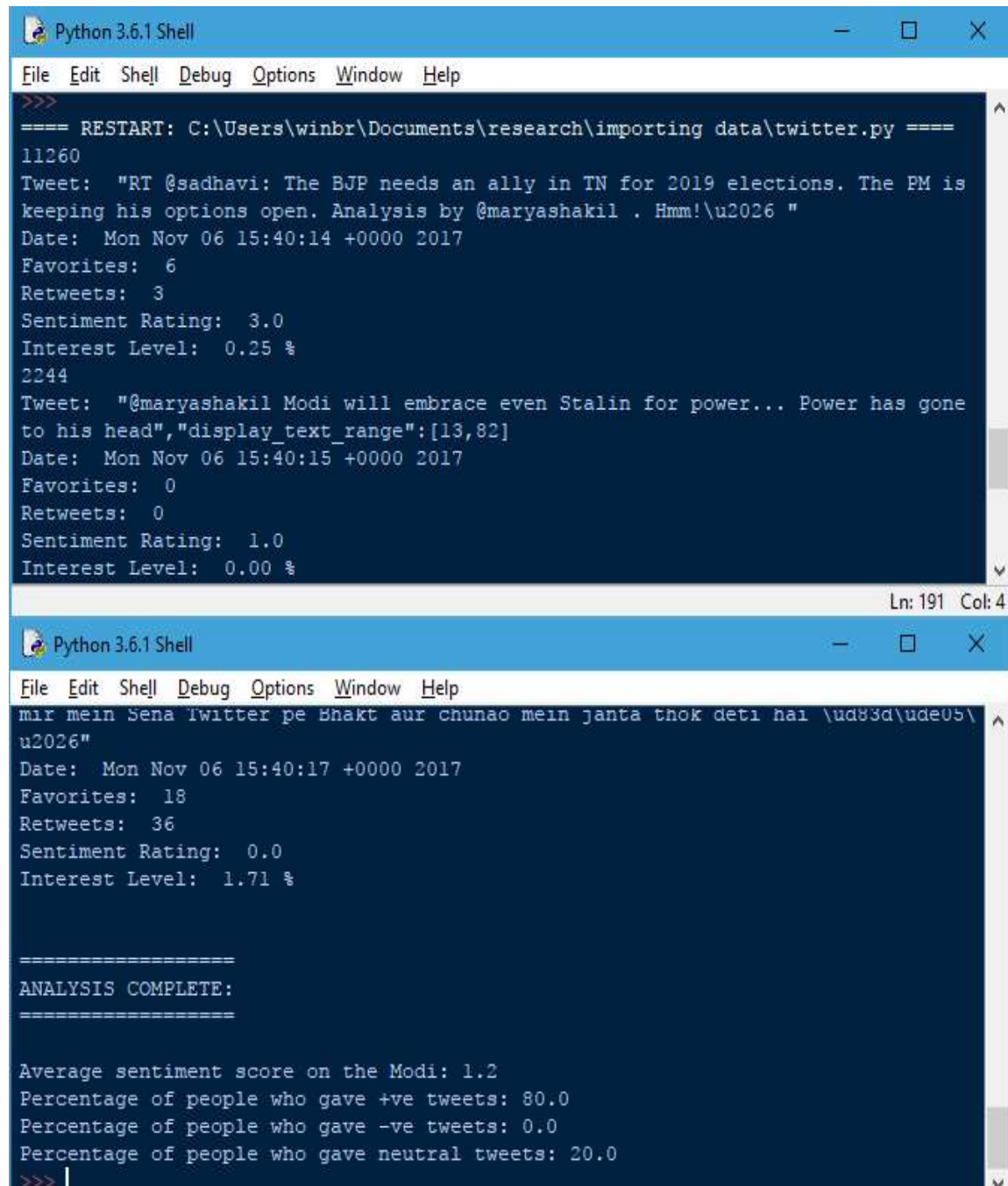
files = [open(path, 'r') for path in dictionary_paths]
dictionaries = [yaml.load(dict_file) for dict_file in files]
map(lambda x: x.close(), files)
self.dictionary = {}
self.max_key_size = 0
for curr_dict in dictionaries:
    for key in curr_dict:
        if key in self.dictionary:
            self.dictionary[key].extend(curr_dict[key])
        else:
            self.dictionary[key] = curr_dict[key]
            self.max_key_size = max(self.max_key_size, len(key))

def tag(self, postagged_sentences):
    return [self.tag_sentence(sentence) for sentence in postagged_sentences]

def tag_sentence(self, sentence, tag_with_lemmas=False):
    """
    the result is only one tagging of all the possible ones.
    The resulting tagging is determined by these two priority rules:
        - longest matches have higher priority
        - search is made from left to right
    """
    tag_sentence = []
    N = len(sentence)
    if self.max_key_size == 0:
        self.max_key_size = N
    i = 0
    while (i < N):
        j = min(i + self.max_key_size, N) #avoid overflow
        tagged = False
        while (j > i):
            expression_form = ' '.join([word[0] for word in sentence[i:j]]).lower()
            expression_lemma = ' '.join([word[1] for word in sentence[i:j]]).lower()
            if tag_with_lemmas:
                literal = expression_lemma
            else:
                literal = expression_form
            if literal in self.dictionary:
                is_single_token = j - i == 1
                #is_single_token = 1
```

Figure 7: Main Code

### 5.1.2.3. OUTPUT



```
>>>
==== RESTART: C:\Users\winbr\Documents\research\importing data\twitter.py ====
11260
Tweet: "RT @sadhavi: The BJP needs an ally in TN for 2019 elections. The PM is
keeping his options open. Analysis by @maryashakil . Hmm!\u2026 "
Date: Mon Nov 06 15:40:14 +0000 2017
Favorites: 6
Retweets: 3
Sentiment Rating: 3.0
Interest Level: 0.25 %
2244
Tweet: "@maryashakil Modi will embrace even Stalin for power... Power has gone
to his head","display_text_range":[13,82]
Date: Mon Nov 06 15:40:15 +0000 2017
Favorites: 0
Retweets: 0
Sentiment Rating: 1.0
Interest Level: 0.00 %
Ln: 191 Col: 4

Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
mir mein Sena Twitter pe Bhakt aur chunao mein janta thok deti hai \ud83d\udef5\
u2026"
Date: Mon Nov 06 15:40:17 +0000 2017
Favorites: 18
Retweets: 36
Sentiment Rating: 0.0
Interest Level: 1.71 %

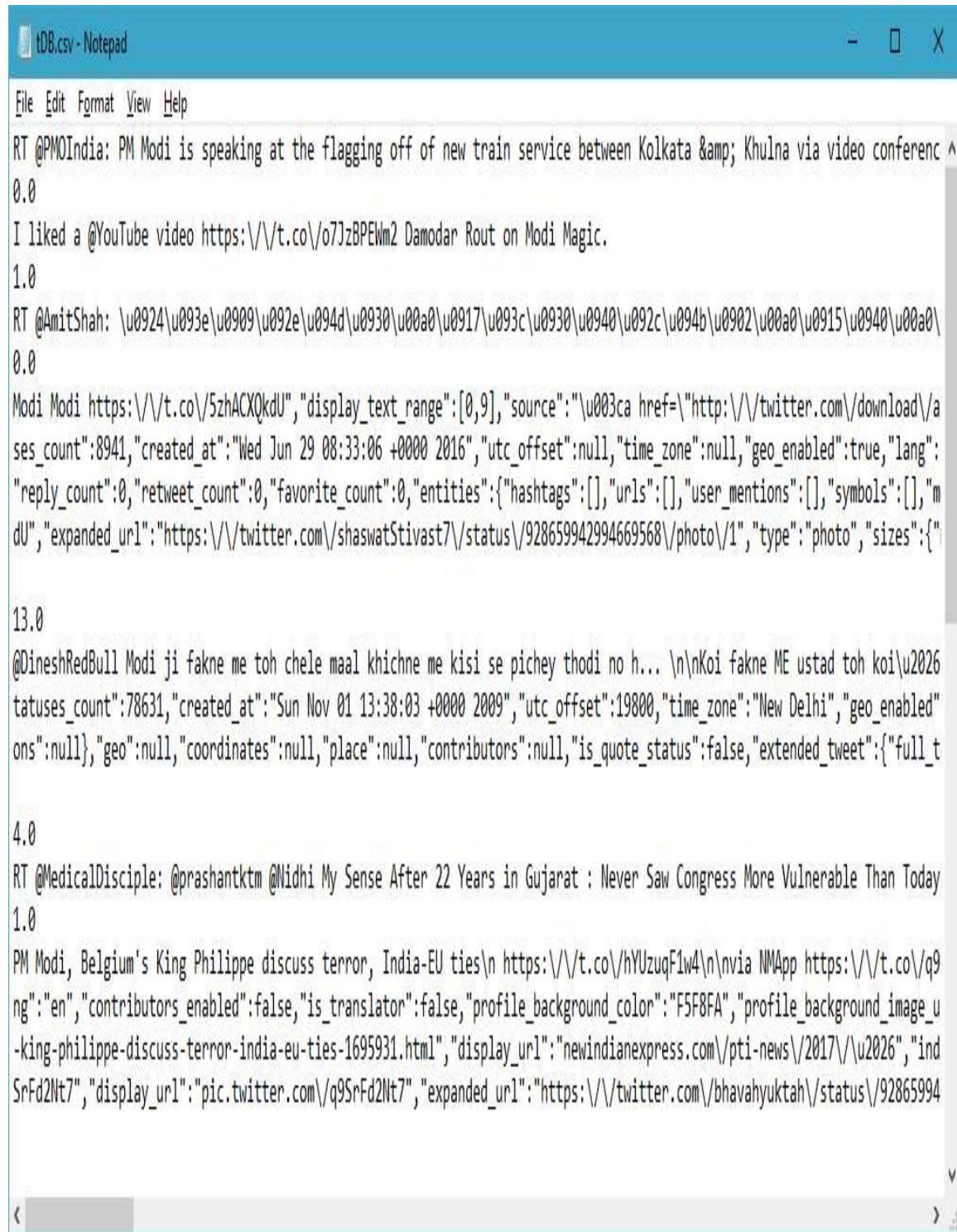
=====
ANALYSIS COMPLETE:
=====

Average sentiment score on the Modi: 1.2
Percentage of people who gave +ve tweets: 80.0
Percentage of people who gave -ve tweets: 0.0
Percentage of people who gave neutral tweets: 20.0
>>>
```

Figure 8: Output



### 5.1.3. DATA SET



```
File Edit Format View Help
RT @PMOIndia: PM Modi is speaking at the flagging off of new train service between Kolkata & Khulna via video conferenc
0.0
I liked a @YouTube video https://t.co/o7JzBPEWm2 Damodar Rout on Modi Magic.
1.0
RT @AmitShah: \u0924\u093e\u0909\u092e\u094d\u0930\u00a0\u0917\u093c\u0930\u0940\u092c\u094b\u0902\u00a0\u0915\u0940\u00a0\
0.0
Modi Modi https://t.co/5zhACXQkdU", "display_text_range": [0, 9], "source": "\u003ca href=\"http://twitter.com/download/a
ses_count": 8941, "created_at": "Wed Jun 29 08:33:06 +0000 2016", "utc_offset": null, "time_zone": null, "geo_enabled": true, "lang":
"reply_count": 0, "retweet_count": 0, "favorite_count": 0, "entities": {"hashtags": [], "urls": [], "user_mentions": [], "symbols": [], "m
dU", "expanded_url": "https://twitter.com/shaswatStivast7/status/928659942994669568/photo/1", "type": "photo", "sizes": {"
13.0
@DineshRedBull Modi ji fakne me toh chele maal khichne me kisi se pichey thodi no h... \n\nKoi fakne ME ustad toh koi\u2026
tatuses_count": 78631, "created_at": "Sun Nov 01 13:38:03 +0000 2009", "utc_offset": 19800, "time_zone": "New Delhi", "geo_enabled"
ons": null}, "geo": null, "coordinates": null, "place": null, "contributors": null, "is_quote_status": false, "extended_tweet": {"full_t
4.0
RT @MedicalDisciple: @prashantktm @Nidhi My Sense After 22 Years in Gujarat : Never Saw Congress More Vulnerable Than Today
1.0
PM Modi, Belgium's King Philippe discuss terror, India-EU ties\n https://t.co/hYUzuqF1w4\n\nvia NMAApp https://t.co/q9
ng": "en", "contributors_enabled": false, "is_translator": false, "profile_background_color": "F5F8FA", "profile_background_image_u
-king-philippe-discuss-terror-india-eu-ties-1695931.html", "display_url": "newindianexpress.com/pti-news/2017/\u2026", "ind
SrFd2Nt7", "display_url": "pic.twitter.com/q9SrFd2Nt7", "expanded_url": "https://twitter.com/bhavahyuktah/status/92865994
```

Figure 9: Dataset

#### 5.1.4. GRAPH ANALYSIS

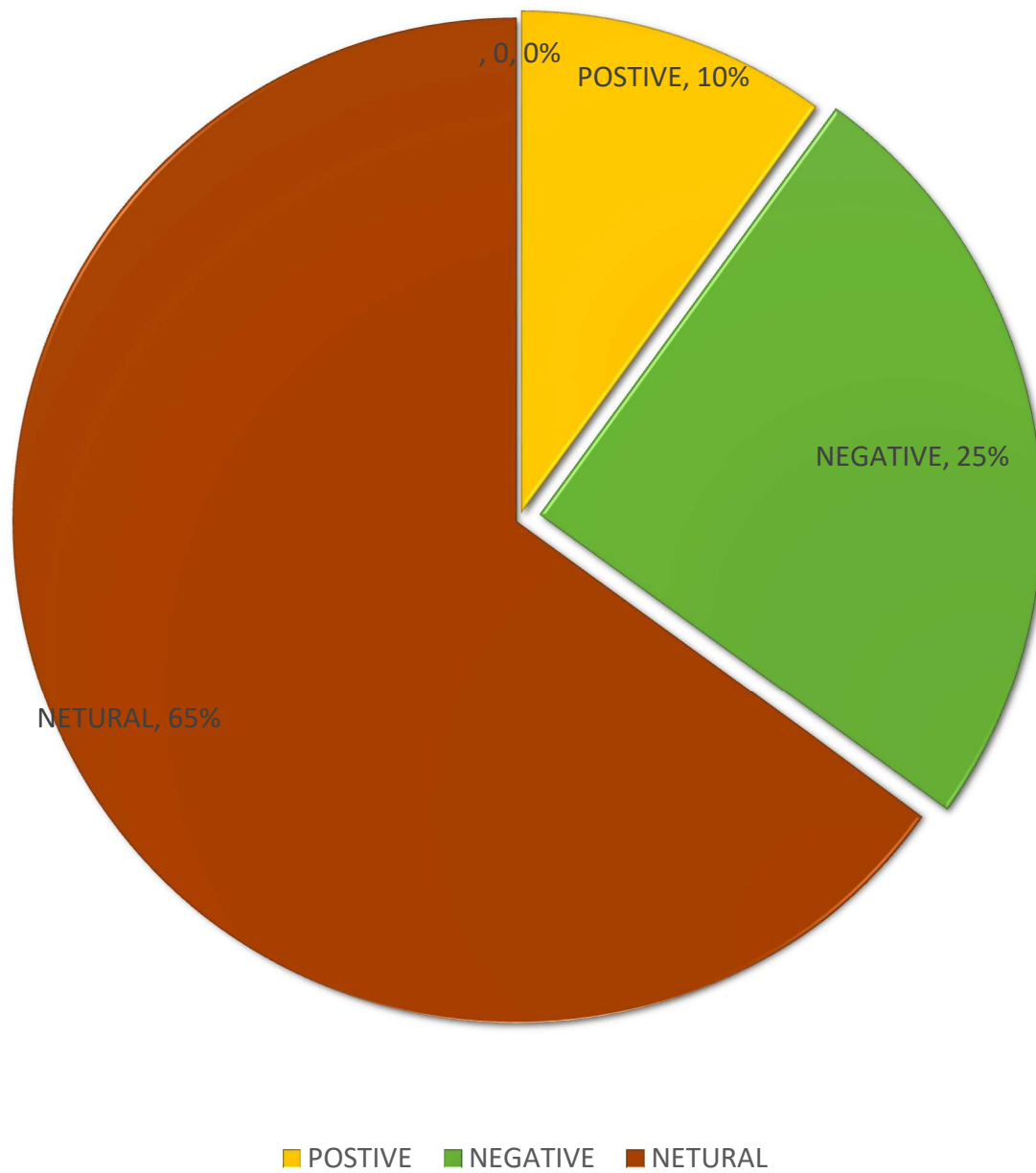


Figure 10: Phase I, Graph Analysis

### **5.1.5. CONCLUSION**

After examining the case study , Average sentiment score on Modi is 0.65. Percentage of people who gave positive tweets are 10.0 while Negative Percentage of Tweets are 25.0. And Percentage of people who gave neutral tweets are 65.0.



## 5.2. PHASE 2

### 5.2.1. CODE

#### KnowledgeBase.py:

```
import nltk
import re
import time
try:
    import urllib.request as urllib2
except ImportError:
    import urllib2
try:
    from urllib.request import urlopen
except ImportError:
    from urllib2 import urlopen
import http.cookiejar as cookielib
from http.cookiejar import CookieJar
import datetime
import sqlite3
import yaml

cj = CookieJar()
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
opener.addheaders = [('User-agent', 'Chrome/61.0')]

conn = sqlite3.connect('knowledgeBase.db')
c = conn.cursor()
site = 'http://www.imdb.com'

contentArray = ['Hey! Have a very nice day.',
                'I am going outside to play football in this weather.',
                'Let\'s get some content from IMDB reviews first shall we?']
ratingArray = ['1', '2', '3']
def loadDicts(dictionary_paths):
    i=0
    files = [open(path, 'r') for path in dictionary_paths]
    dictionaries = [yaml.load(dict_file) for dict_file in files]
    map(lambda x: x.close(), files)
    dictionary = {}
    for curr_dict in dictionaries:
        for key in curr_dict:
            if key not in dictionary:
                print(key)
                print(curr_dict[key])
                if curr_dict[key][0] == 'positive':
                    dictionary[key] = 10
```

```

        elif curr_dict[key][0] == 'negative':
            dictionary[key] = -10
        elif curr_dict[key][0] == 'dec':
            dictionary[key] = 0.5
        elif curr_dict[key][0] == 'inc':
            dictionary[key] = 2.0
        elif curr_dict[key][0] == 'inv':
            dictionary[key] = -1.0

    return dictionary

def IMDBContent():
    try:
        page = 'http://www.imdb.com/?ref=nv_home'
        sourceCode = opener.open(page).read().decode('utf-8')
        try:
            #links = re.findall(r'<a.*href=\"(.*)\"', sourceCode)
            links = []
            try:
                for i in range(1,100):
links.append(sourceCode.split('<div
href=""')[i].split('>')[0])
                    except :
                        pass
                    print(links)
                    reviewLinks = []
                    for link in links:
                        try:
                            subpage = site+link
                            linkCode = opener.open(subpage).read().decode('utf-
8')

                            tID = link[7:16]
                            print(tID)
                            for start in {0,10,20,30}:
                                reviewLink =
'http://www.imdb.com/title/'+tID+'/reviews?start='+str(start)
                                print(reviewLink)
                                reviewLinks.append(reviewLink)
                        except Exception as e:
                            print('Failed in the 3rd loop')
                            print(str(e))
                    print(reviewLinks)
                    for link in reviewLinks:
                        try:
                            for i in range(1,10):
                                fail = False
                                try:
                                    linkCode =
opener.open(link).read().decode('utf-8')
                                    reviewStart = '<img width="102" height="12"
alt=""

                                    reviewEnd = '</p>\n\n<div class="yn"'
                                    review =
linkCode.split(reviewStart)[i].split(reviewEnd)[0]
                                    review = '::rate::'+review+'::content::'
                                    rating =
review.split('::rate::')[1].split('/10')[0]

```

```

        content
review.split('</div>\n<p>')[1].split('::content::')[0]
        print(rating)
        print(content)
    except Exception as e:
        print('Failed in 4th loop, ', str(e))
        fail = True
    if not fail:
        ratingArray.append(rating)
        contentArray.append(content)

    except Exception as e:
        print('Failed in 3rd b loop, ', str(e))
except Exception as e:
    print('Failed in the 2nd loop')
    print (str(e))

except Exception as e:
    print ('Failed in the 1st loop')
    print (str(e))

i = 1
for content in contentArray:
    print('Rating: '+str(i))
    print(ratingArray[i-1])
    if len(content)>0:
        print('Content: '+str(i),',', Size: '+str(len(content)))
    else:
        print('Content: '+str(i),'None')
    print(content)
    i+=1

def buildWordBase(dictionary):
    for key in dictionary:
        currentTime = time.time()
        value = dictionary[key]
        dateStamp
datetime.datetime.fromtimestamp(currentTime).strftime('%Y-%m-%d %H: %M: %S')
        c.execute("INSERT INTO wordBase (dateStamp, word, polarity,
isNeutral) VALUES (?, ?, ?, ?);",
                    (dateStamp, key, value, 0))
        conn.commit()

def buildMultiplierBase(dictionary):
    for key in dictionary:
        currentTime = time.time()
        value = dictionary[key]
        dateStamp
datetime.datetime.fromtimestamp(currentTime).strftime('%Y-%m-%d %H: %M: %S')
        c.execute("INSERT INTO multiplierBase (dateStamp, adverb,
factor, isNeutral) VALUES (?, ?, ?, ?);",
                    (dateStamp, key, value, 0))
        conn.commit()

```

```

dict_paths      =      [      'dicts/pos_dict.yml',      'dicts/newpos.yml',
'dicts/pos.yml', 'dicts/pos2.yml',
                        'dicts/neg_dict.yml', 'dicts/newneg.yml']
multiplier_dict = ['dicts/inc.yml', 'dicts/dec.yml', 'dicts/inv.yml']
#dictionary = loadDicts(multiplier_dict)
#buildMultiplierBase(dictionary)
IMDBcontent()
site = 'https://www.amazon.in'
aSearch      =      'https://www.amazon.in/s/ref=nb_sb_noss_2?url=search-
alias%3Daps&field-keywords='
keyword = 'phones'
def amazonContent():
    try:
        page = aSearch+keyword
        sourceCode = opener.open(page).read().decode('utf-8')
        try:
            #links = re.findall(r'<a.*href=\"(.*)\"', sourceCode)
            links = []
            opening = '<div class="a-row a-spacing-none"><a class="a-
link-normal a-text-normal" href="'
            closing      =      '><span      class="a-size-small      a-color-
secondary"></span><span      class="a-size-base      a-color-price      s-price      a-
text-bold"><span class="currencyINR">'
            try:
                for          i          in          range(1,100):
links.append(sourceCode.split(opening)[i].split(closing)[0])
            except :
                pass
            print(links)
            reviewLinks = []
            for link in links:
                try:
                    linkCode = opener.open(link).read().decode('utf-8')
                    opening = '<a id="acrCustomerReviewLink" class="a-
link-normal" href="'
                    closing = '>'
                    reviewLink
                    =
site+linkCode.split(opening)[1].split(closing)[0]
                    print(reviewLink)
                    reviewLinks.append(reviewLink)
                except Exception as e:
                    print('Failed in the 3rd loop')
                    print(str(e))
            print(reviewLinks)
            for link in reviewLinks:
                try:
                    for i in range(1,5):
                        linkCode
                        =
opener.open(link).read().decode('utf-8')
                        opening      =      'class="a-size-base      a-link-normal
review-title a-color-base a-text-bold" href="'
                        closing = '</a>'
                        rClosing = ' out of 5 stars'
                        rating      =      linkCode.split('review-rating"><span
class="a-icon-alt">')[i].split(closing)[0]
                        rating = '::start::'+rating

```

```

        testR
        rating.split(opening)[1].split(closing)[0]
        testR+='::end::'
        testR
    testR.split('">')[1].split('::end::')[0]
    rating
    rating.split('::start::')[1].split(rClosing)[0]
    print(testR)
    print(rating)
    ##
    ##
    ratingArray.append(rating)
    contentArray.append(testR)
except Exception as e:
    print('Failed in 3rd b loop: ', str(e))
except Exception as e:
    print('Failed in the 2nd loop')
    print (str(e))

except Exception as e:
    print ('Failed in the 1st loop')
    print (str(e))

#amazonContent()

def buildBase(polarity, content):
    tokenized = nltk.word_tokenize(content)
    postagged = nltk.pos_tag(tokenized)
    for word in postagged:
        polarity2 = polarity
        if word[1][:2] == 'JJ':
            row = c.execute("SELECT * FROM wordBase;")
            exists = False
            currentTime = time.time()
            dateStamp
            datetime.datetime.fromtimestamp(currentTime).strftime('%Y-%m-%d %H: %M: %S')
            for eachRow in row:
                if word[0].lower() == eachRow[1]:
                    print(eachRow)
                    print(word[0])
                    exists = True
                    pol = c.execute("SELECT polarity FROM wordBase
WHERE word=?", (word[0],))
                    for eachPol in pol:
                        print(eachPol)
                        polarity2+=eachPol[0]
                    c.execute("UPDATE wordBase SET dateStamp=?, word=?,
polarity=?, isNeutral=? WHERE word=?",
                            (dateStamp, eachRow[1], polarity2, 0,
word[0].lower()));
                    if not exists:
                        c.execute("INSERT INTO wordBase VALUES (dateStamp,
word, polarity, isNeutral) VALUES (?, ?, ?, ?);",
                                (dateStamp, word[0], polarity2, 0))
                        print(polarity2)
                        conn.commit()
                    elif word[1] == 'RB':

```

```

        row = c.execute("SELECT * FROM multiplierBase;")
        if polarity>0:
            polarity2 = 2.0
        elif polarity<0:
            polarity2 = 0.5
        exists = False
        currentTime = time.time()
        dateStamp =
datetime.datetime.fromtimestamp(currentTime).strftime('%Y-%m-%d %H: %M:
%S')
        for eachRow in row:
            if word[0].lower() == eachRow[1]:
                print(eachRow)
                print(word[0])
                exists = True
                pol = c.execute("SELECT factor FROM multiplierBase
WHERE adverb=?", (word[0],))
                for eachPol in pol:
                    print(eachPol)
                    polarity2*=eachPol[0]
                c.execute("UPDATE multiplierBase SET dateStamp=?,
adverb=?, factor=?, isNeutral=? WHERE adverb=?",
                        (dateStamp, eachRow[1], polarity2, 0,
word[0].lower()));
                if not exists:
                    c.execute("INSERT INTO wordBase VALUES (dateStamp,
adverb, factor, isNeutral) VALUES (?, ?, ?, ?);",
                        (dateStamp, word[0], polarity2, 0))
                print(polarity2)
                conn.commit()
def processContent():
    try:
        i = 1
        for content in contentArray:
            if len(content)>0 and len(content)<2000:
                if int(ratingArray[i-1])>6:
                    buildBase(1, content)
                elif int(ratingArray[i-1])<5:
                    buildBase(-1, content)
                i+=1
            except Exception as e:
                print ('Error: ',e)

```

## processContent()

---

```

features.py
import re
import json

```

```

class Emoticons:
    def analyse(self, string):

        self.string = re.sub(
            r'\W+:\)\(\'\{\}\}\-\@>\<=\;\[\\]!\',

```

```

        ' ',
        string)
self.string = self.string.replace('.', '')
self.string = self.string.replace('?', '')
self.words = self.string.split(" ")
if self.words[-1] == ':':
    del self.words[-1]
positiveEmoz = [
    ':)',
    ':-)',
    ':D',
    ':-D',
    ':P',
    ':p',
    ':-P',
    ';)',
    ';-)',
    ';D',
    ';-D',
    ':o)',
    ':]',
    ':3',
    ':c)',
    ':>',
    '=]',
    '8)',
    'B)',
    'BD',
    '<3',
    '=)',
    ':}',
    '8D',
    'xD',
    'XD',
    'X-D',
    '=D',
    '=3',
    ':-) )',
    ':\')',
    'lol',
    'lol!']
negativeEmoz = [
    ':(',
    ':-(',
    ':(',
    ':-(',
    ':-<',
    ':-[',
    ':[',
    ':{',
    ':-||',
    ':@',
    ':\'-(',
    ':\'(',
    'QQ',
    'D:',
    'D:<',

```

```

        'D8',
        'D;',
        'DX',
        '</3',
        '<\\3',
        'v.v',
        '>.<',
        'D=']
    positiveCount = 0
    negativeCount = 0
    for i in self.words:
        if i in positiveEmoz:
            positiveCount += 1
        if i in negativeEmoz:
            negativeCount += 1
    positiveEmoz, negativeEmoz = 0, 0
    if positiveCount + negativeCount == 0:
        return {'positive': 0, 'negative': 0}
    return {'positive': positiveCount, 'negative': negativeCount}

def repairString(string):
    data = {
        'm': 'am',
        'u': 'you',
        'ua': 'your',
        'yrs': 'years',
        'ur': 'your',
        'urs': 'yours',
        'tc': 'take care',
        'gn': 'good night',
        'gm': 'good morning',
        'ryt': 'right',
        'nite': 'night',
        'wat': 'what',
        'abt': 'about',
        'k': 'okay',
        'knw': 'know',
        'nt': 'not',
        'w8': 'wait',
        'f9': 'fine',
        'wbu': 'what about you',
        'kk': 'okay',
        'ok': 'okay',
        'na': 'no',
        'don\\t': 'do not',
        'won\\t': 'will not',
        'gonna': 'going to',
        'juz': 'just',
        'jus': 'just',
        'fk': 'fuck',
        'wtf': 'what the fuck',
        'shud': 'should',
        'coz': 'because',
        'cos': 'because',
        'ttyl': 'talk to you later',
        'ty': 'thank you',

```



```

        'hlo': 'hello',
        'helo': 'hello',
        'hola': 'hello',
        '&#x27;': '\'',
        'wut': 'what',
        'gtfo': 'get the fuck out',
        'whr': 'where',
        'y': 'why',
        'ohk': 'okay'}
string = string.split(" ")
for i in string:
    if i.lower() in data:
        index = string.index(i)
        string[index] = data[i.lower()]
return " ".join(string)

```

### **sentiment\_analyzer.py:**

```

import nltk, yaml
from emo import split_emo
import datetime
import sqlite3
import features

class Splitter(object):
    def __init__(self):
        self.nltk_splitter =
nltk.data.load('tokenizers/punkt/english.pickle')
        self.nltk_tokenizer = nltk.tokenize.TreebankWordTokenizer()

    def split(self, text):
        sentences = self.nltk_splitter.tokenize(text)
        tokenized_sentences = [self.nltk_tokenizer.tokenize(sent) for
sent in sentences]
        tokenized_sentences = split_emo(tokenized_sentences)
        return tokenized_sentences

class POSTagger(object):
    def __init__(self):
        pass

    def pos_tag(self, sentences):

        pos = [nltk.pos_tag(sentence) for sentence in sentences]
        #adapt format
        pos = [[(word, word, [postag]) for (word, postag) in sentence]
for sentence in pos]
        return pos

class DictionaryTagger(object):
    def __init__(self):
        self.dictionary = {}

```

```

self.max_key_size = 0
conn = sqlite3.connect('knowledgeBase.db')
c = conn.cursor()
row = c.execute("SELECT * FROM wordBase;")
for eachRow in row:
    if eachRow[1] in self.dictionary:
        pass
    else:
        self.max_key_size = max(self.max_key_size,
len(eachRow[1]))
        if int(eachRow[3])==1:
            self.dictionary[eachRow[1]] = ['neutral']
        elif int(eachRow[2])>0:
            self.dictionary[eachRow[1]] = ['positive']
        elif int(eachRow[2])<0:
            self.dictionary[eachRow[1]] = ['negative']

row = c.execute("SELECT * FROM multiplierBase;")
for eachRow in row:
    if eachRow[1] in self.dictionary:
        pass
    else:
        self.max_key_size = max(self.max_key_size,
len(eachRow[1]))
        if int(eachRow[3])==1:
            self.dictionary[eachRow[1]] = ['neutral']
        elif int(eachRow[2])>1:
            self.dictionary[eachRow[1]] = ['inc']
        elif int(eachRow[2])<1 and int(eachRow[2])>0:
            self.dictionary[eachRow[1]] = ['dec']
        elif int(eachRow[2])<0:
            self.dictionary[eachRow[1]] = ['inv']

def tag(self, postagged_sentences):
    return [self.tag_sentence(sentence) for sentence in
postagged_sentences]

def tag_sentence(self, sentence, tag_with_lemmas=False):
    tag_sentence = []
    N = len(sentence)
    if self.max_key_size == 0:
        self.max_key_size = N
    i = 0
    while (i < N):
        j = min(i + self.max_key_size, N) #avoid overflow
        tagged = False
        while (j > i):
            expression_form = ' '.join([word[0] for word in
sentence[i:j]]).lower()
            expression_lemma = ' '.join([word[1] for word in
sentence[i:j]]).lower()
            if tag_with_lemmas:
                literal = expression_lemma
            else:
                literal = expression_form

```

```

        if literal in self.dictionary:
            is_single_token = j - i == 1
            original_position = i
            i = j
            taggings = [tag for tag in
self.dictionary[literal]]
            tagged_expression = (expression_form,
expression_lemma, taggings)
            if is_single_token:
                original_token_tagging =
sentence[original_position][2]
tagged_expression[2].extend(original_token_tagging)
            tag_sentence.append(tagged_expression)
            tagged = True
        else:
            j = j - 1
        if not tagged:
            tag_sentence.append(sentence[i])
            i += 1
    return tag_sentence

```

```

def value_of(sentiment):
    if sentiment == 'positive': return 1
    if sentiment == 'negative': return -1
    return 0

```

```

def sentence_score(sentence_tokens, previous_token, acum_score):
    if not sentence_tokens:
        return acum_score
    else:
        current_token = sentence_tokens[0]
        tags = current_token[2]
        token_score = sum([value_of(tag) for tag in tags])
        if previous_token is not None:
            previous_tags = previous_token[2]
            prev_token_score = sum([value_of(tag) for tag in
previous_token[2]])
            if 'inc' in previous_tags:
                token_score *= 2.0
            elif 'dec' in previous_tags:
                token_score /= 2.0
            elif 'inv' in previous_tags:
                token_score *= -1.0
        return sentence_score(sentence_tokens[1:], current_token,
acum_score + token_score)

```

```

def sentiment_score(review):
    return sum([sentence_score(sentence, None, 0.0) for sentence in
review])

```

```

##text = """What can I say about this place. :-( The staff of the
restaurant is nice and the eggplant is not bad. Apart from that, very
uninspired food, lack of atmosphere and too expensive. :) I am a
staunch vegetarian and was sorely dissapointed with the veggie options

```

```

on the menu. Will be the last time I visit, I recommend others to
avoid."""
##text = ""He killed him! Killed him. Great! :( Wtf are you doing?""
##splitter = Splitter()
##postagger = POSTagger()
##emoji = features.Emoticons()
##emos = emoji.analyse(text)
##text = features.repairString(text)
##print(emos)
##splitted_sentences = splitter.split(text)
##
##print(splitted_sentences)
##print('\n\n')
##pos_tagged_sentences = postagger.pos_tag(splitted_sentences)
##
##print(pos_tagged_sentences)
##
##
##dicttagger = DictionaryTagger(['dicts/pos_dict.yml',
'dicts/newpos.yml', 'dicts/pos.yml', 'dicts/pos2.yml',
'dicts/neg_dict.yml', 'dicts/newneg.yml', 'dicts/inc.yml',
'dicts/dec.yml', 'dicts/inv.yml'])
##
##dict_tagged_sentences = dicttagger.tag(pos_tagged_sentences)
##
##print(dict_tagged_sentences)
##sentiments = sentiment_score(dict_tagged_sentences) +
emos['positive'] - emos['negative']
##print(sentiments)

```

---

## Twitter.py:

```

from tweepy import Stream, OAuthHandler
from tweepy.streaming import StreamListener
import time, urllib, re
from textblob import TextBlob
from sentiment_analyser import Splitter, POSTagger, DictionaryTagger,
value_of
from sentiment_analyser import sentence_score, sentiment_score
import features

ckey = 'McfEpmGoArTagly0YEgLmAw9B'
csecret = 'nhh8YrANBISlCcjm6T1XZr9dDuBOFBK69fyR3vIISlNKQUzmTF'
atoken = '580508392-TGvrelm0EqXppjxZoiNoB7B4bfDBxG6lm4rOwIdo'
asecret = 'ZWLk0U1KilShIZHeJnYGshMhxI11JLHL9WurC9IsNck0m'

def get_sentiment(tweet):
    analysis = TextBlob(tweet)
    return analysis.sentiment.polarity

class listener(StreamListener):
    def __init__(self):
        self.limit = 20
        self.count = 0

```

```

self.positive_score = 0.0
self.p_count = 0.0
self.n_count = 0.0
self.neutral = 0.0
self.negative_score = 0.0

def on_data(self, data):
    self.count+=1
    if self.count>self.limit:
        print('\n\n=====')
        print('ANALYSIS COMPLETE: ')
        print('=====\n')
        print('Average      sentiment      score      on      the      Modi:
'+str((self.positive_score+self.negative_score)/self.limit))
        print('Percentage of people who gave +ve tweets:
'+str((100*self.p_count/self.limit)))
        print('Percentage of people who gave -ve tweets:
'+str((100*self.n_count/self.limit)))
        print('Percentage of people who gave neutral tweets:
'+str((100*self.neutral/self.limit)))
        exit(0)
    tweet = data.split(', "text":') [1].split(', "source":') [0]
    text = str(tweet)
    splitter = Splitter()
    postagger = POSTagger()
    emoji = features.Emoticons()
    emos = emoji.analyse(text)
    text = features.repairString(text)
    splitted_sentences = splitter.split(text)
    pos_tagged_sentences = postagger.pos_tag(splitted_sentences)
    dicttagger = DictionaryTagger()
    dict_tagged_sentences = dicttagger.tag(pos_tagged_sentences)
    senti = sentiment_score(dict_tagged_sentences) +
emos['positive'] - emos['negative']
    if senti>0:
        self.positive_score+=senti
        self.p_count+=1
    elif senti<0:
        self.negative_score-=senti
        self.n_count+=1
    else:
        self.neutral+=1
    date = data.split('{ "created_at":') [1].split(', "id":') [0]
    favorites =
int(data.split(', "favorite_count":') [1].split(', "entities":') [0])
    followers =
int(data.split(', "followers_count":') [1].split(', "friends_count":') [0])
    if followers!=0:
        interest_level = 100*favorites/followers
    else:
        interest_level = 0
    retweets =
data.split(', "retweet_count":') [1].split(', "favorite_count":') [0]
    try:
        file = open('tDB.csv', 'a')
        file.write('Feed: \n')
        file.write(data)

```

```

        file.write('\n')
        file.close()
        print('Tweet: ', text, '\nDate: ', date, '\nFavorites: ',
favorites, '\nRetweets: ', retweets, '\nSentiment Rating: ', senti,
'\nInterest Level: ', "%.2f"%interest_level,'%')
    except BaseException:
        print('Failed')
        time.sleep(5)
        return False
    return True

def on_error(self, status):
    print(status)

auth = OAuthHandler(ckey, csecret)
auth.set_access_token(accessToken, asecret)
l = listener()
twitterStream = Stream(auth, l)
twitterStream.filter(track=['Modi'])

```

## 5.2.2. OUTPUTS

### 5.2.2.1. WORD DATABASE

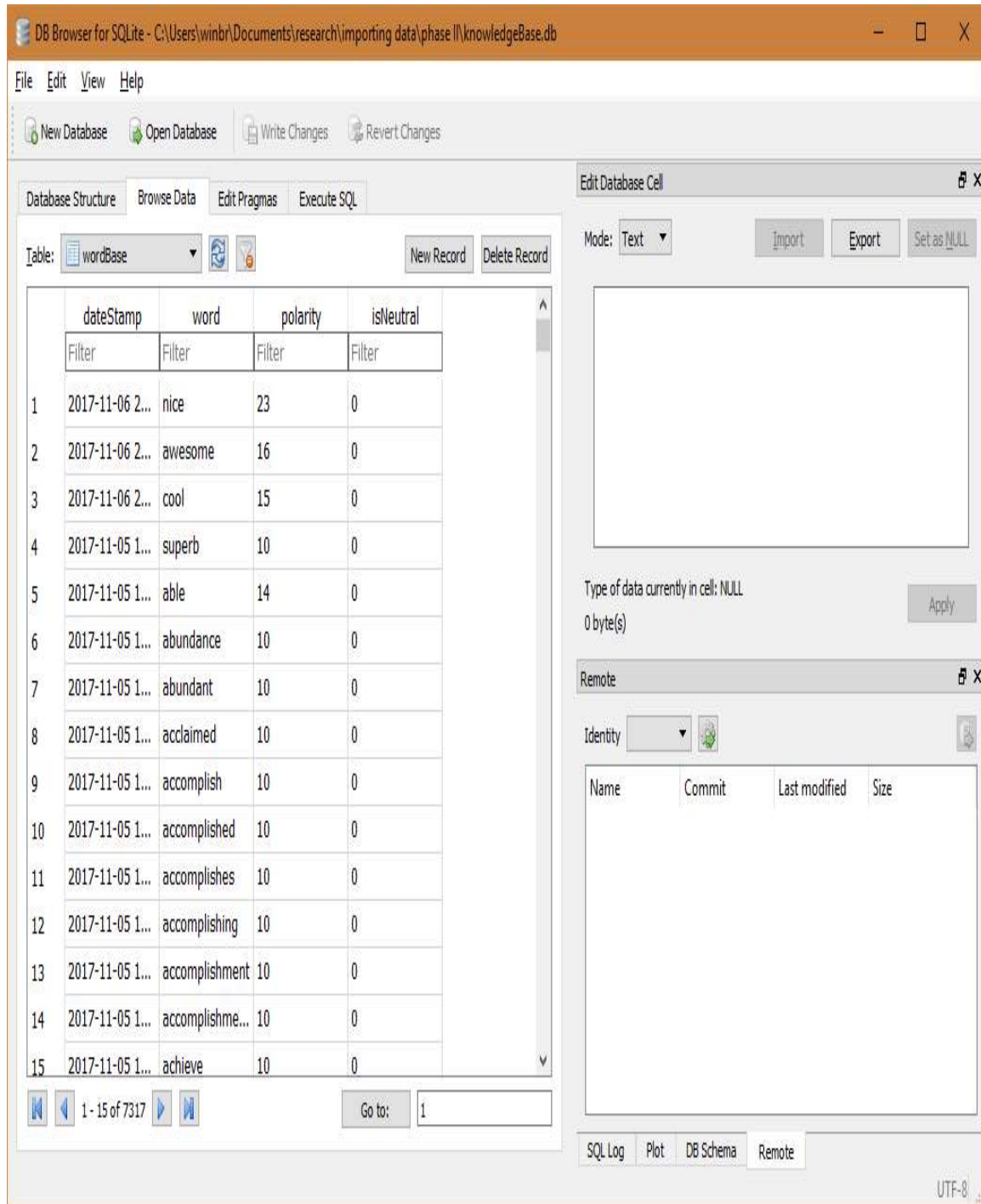
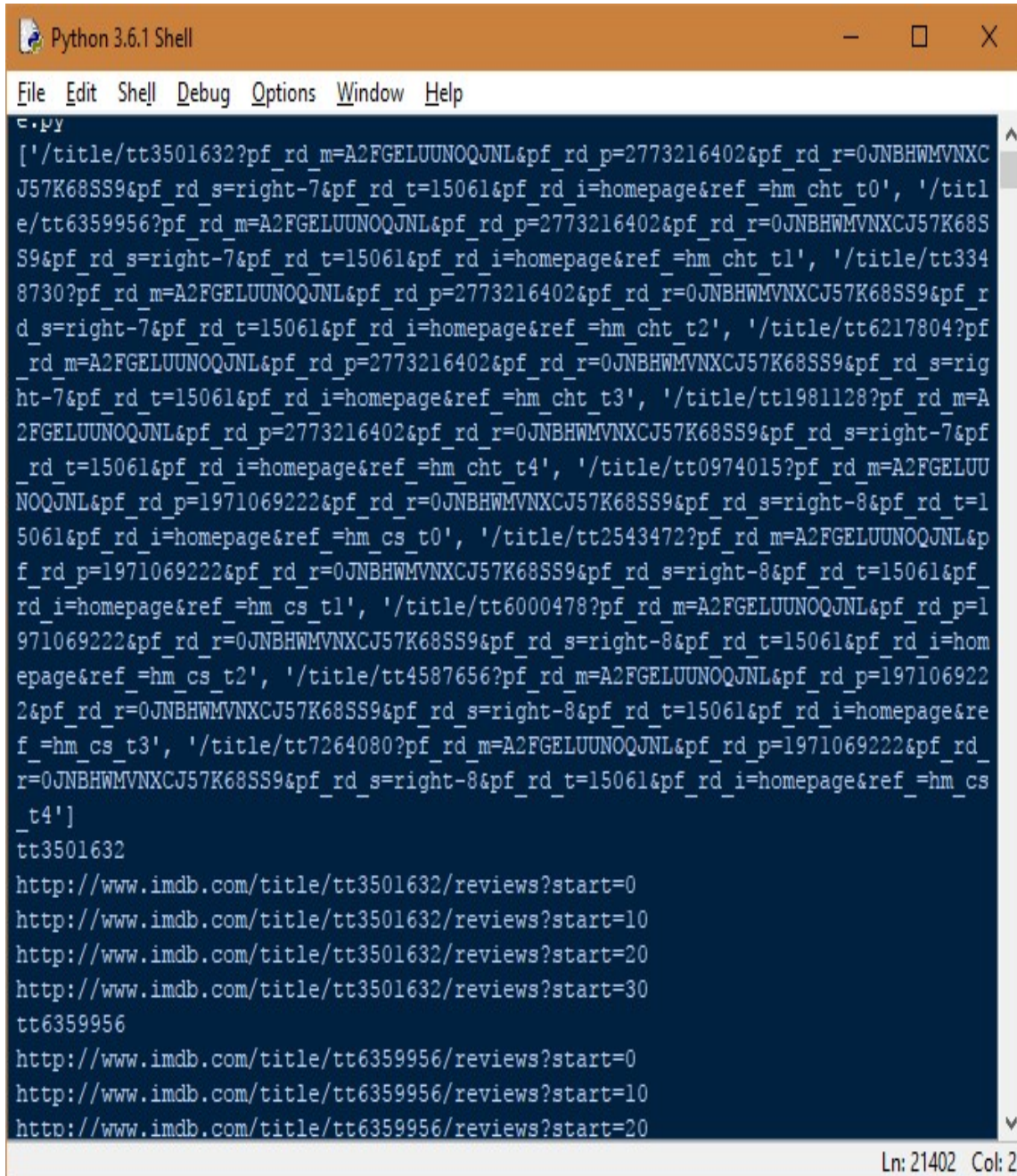


Figure 11: Word Database

### 5.2.2.2. REVIEW LINKS

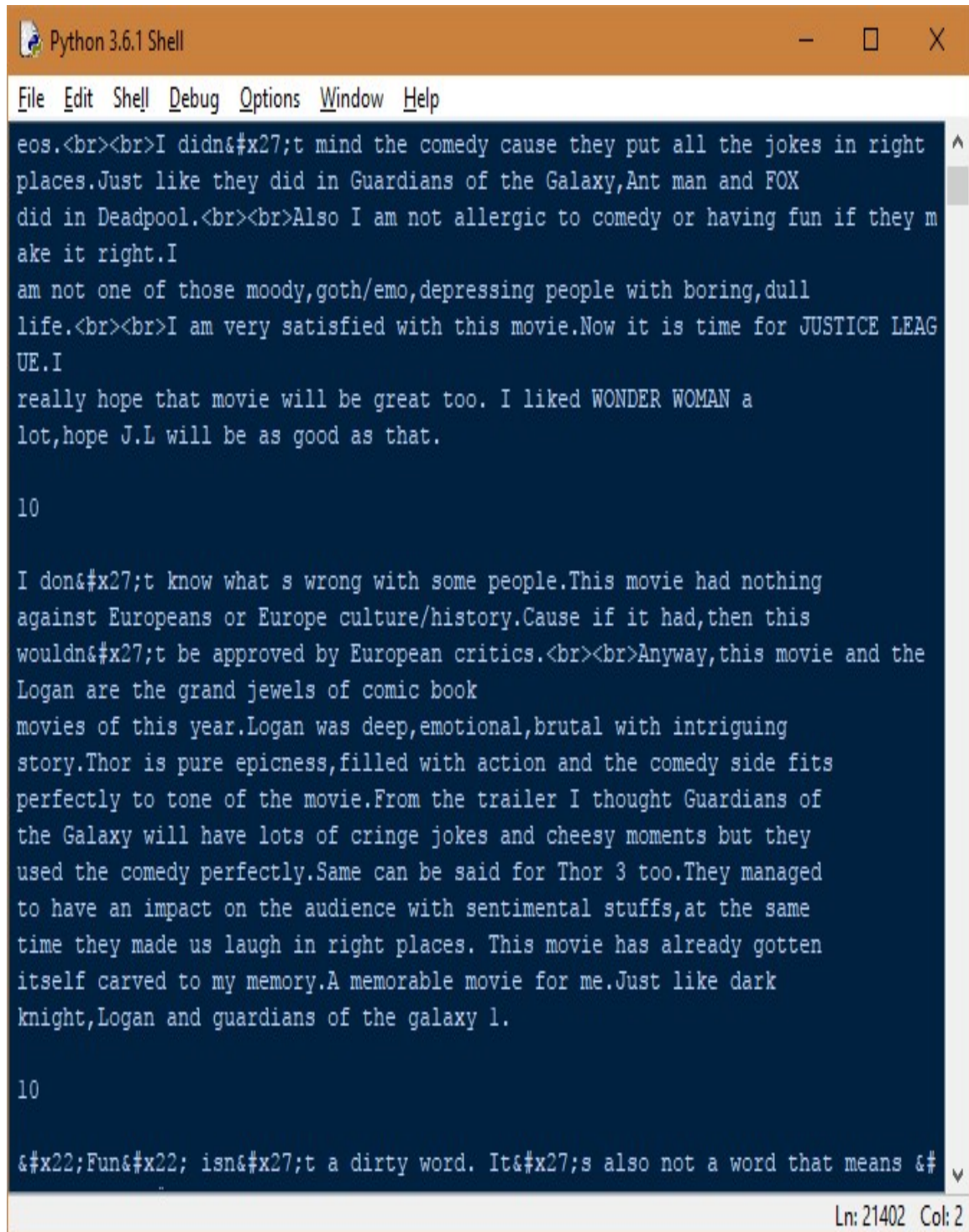


```
Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
c.py
['/title/tt3501632?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=2773216402&pf_rd_r=0JNBHWMVNXC
J57K68SS9&pf_rd_s=right-7&pf_rd_t=15061&pf_rd_i=homepage&ref=hm_chn_t0', '/titl
e/tt6359956?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=2773216402&pf_rd_r=0JNBHWMVNXCJ57K68S
S9&pf_rd_s=right-7&pf_rd_t=15061&pf_rd_i=homepage&ref=hm_chn_t1', '/title/tt334
8730?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=2773216402&pf_rd_r=0JNBHWMVNXCJ57K68SS9&pf_r
d_s=right-7&pf_rd_t=15061&pf_rd_i=homepage&ref=hm_chn_t2', '/title/tt6217804?pf
_rd_m=A2FGELUUNOQJNL&pf_rd_p=2773216402&pf_rd_r=0JNBHWMVNXCJ57K68SS9&pf_rd_s=rig
ht-7&pf_rd_t=15061&pf_rd_i=homepage&ref=hm_chn_t3', '/title/tt1981128?pf_rd_m=A
2FGELUUNOQJNL&pf_rd_p=2773216402&pf_rd_r=0JNBHWMVNXCJ57K68SS9&pf_rd_s=right-7&pf
_rd_t=15061&pf_rd_i=homepage&ref=hm_chn_t4', '/title/tt0974015?pf_rd_m=A2FGELUU
NOQJNL&pf_rd_p=1971069222&pf_rd_r=0JNBHWMVNXCJ57K68SS9&pf_rd_s=right-8&pf_rd_t=1
5061&pf_rd_i=homepage&ref=hm_cs_t0', '/title/tt2543472?pf_rd_m=A2FGELUUNOQJNL&p
f_rd_p=1971069222&pf_rd_r=0JNBHWMVNXCJ57K68SS9&pf_rd_s=right-8&pf_rd_t=15061&pf_
rd_i=homepage&ref=hm_cs_t1', '/title/tt6000478?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=1
971069222&pf_rd_r=0JNBHWMVNXCJ57K68SS9&pf_rd_s=right-8&pf_rd_t=15061&pf_rd_i=hom
epage&ref=hm_cs_t2', '/title/tt4587656?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=197106922
2&pf_rd_r=0JNBHWMVNXCJ57K68SS9&pf_rd_s=right-8&pf_rd_t=15061&pf_rd_i=homepage&re
f=hm_cs_t3', '/title/tt7264080?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=1971069222&pf_rd_
r=0JNBHWMVNXCJ57K68SS9&pf_rd_s=right-8&pf_rd_t=15061&pf_rd_i=homepage&ref=hm_cs
_t4']
tt3501632
http://www.imdb.com/title/tt3501632/reviews?start=0
http://www.imdb.com/title/tt3501632/reviews?start=10
http://www.imdb.com/title/tt3501632/reviews?start=20
http://www.imdb.com/title/tt3501632/reviews?start=30
tt6359956
http://www.imdb.com/title/tt6359956/reviews?start=0
http://www.imdb.com/title/tt6359956/reviews?start=10
http://www.imdb.com/title/tt6359956/reviews?start=20
Ln: 21402 Col: 2
```

Figure 11: Review Links



### 5.2.2.3. REVIEW RATING



```
Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
eos.<br><br>I didn't mind the comedy cause they put all the jokes in right
places.Just like they did in Guardians of the Galaxy,Ant man and FOX
did in Deadpool.<br><br>Also I am not allergic to comedy or having fun if they m
ake it right.I
am not one of those moody,goth/emo,depressing people with boring,dull
life.<br><br>I am very satisfied with this movie.Now it is time for JUSTICE LEAG
UE.I
really hope that movie will be great too. I liked WONDER WOMAN a
lot,hope J.L will be as good as that.

10

I don't know what s wrong with some people.This movie had nothing
against Europeans or Europe culture/history.Cause if it had,then this
wouldn't be approved by European critics.<br><br>Anyway,this movie and the
Logan are the grand jewels of comic book
movies of this year.Logan was deep,emotional,brutal with intriguing
story.Thor is pure epicness,filled with action and the comedy side fits
perfectly to tone of the movie.From the trailer I thought Guardians of
the Galaxy will have lots of cringe jokes and cheesy moments but they
used the comedy perfectly.Same can be said for Thor 3 too.They managed
to have an impact on the audience with sentimental stuffs,at the same
time they made us laugh in right places. This movie has already gotten
itself carved to my memory.A memorable movie for me.Just like dark
knight,Logan and guardians of the galaxy 1.

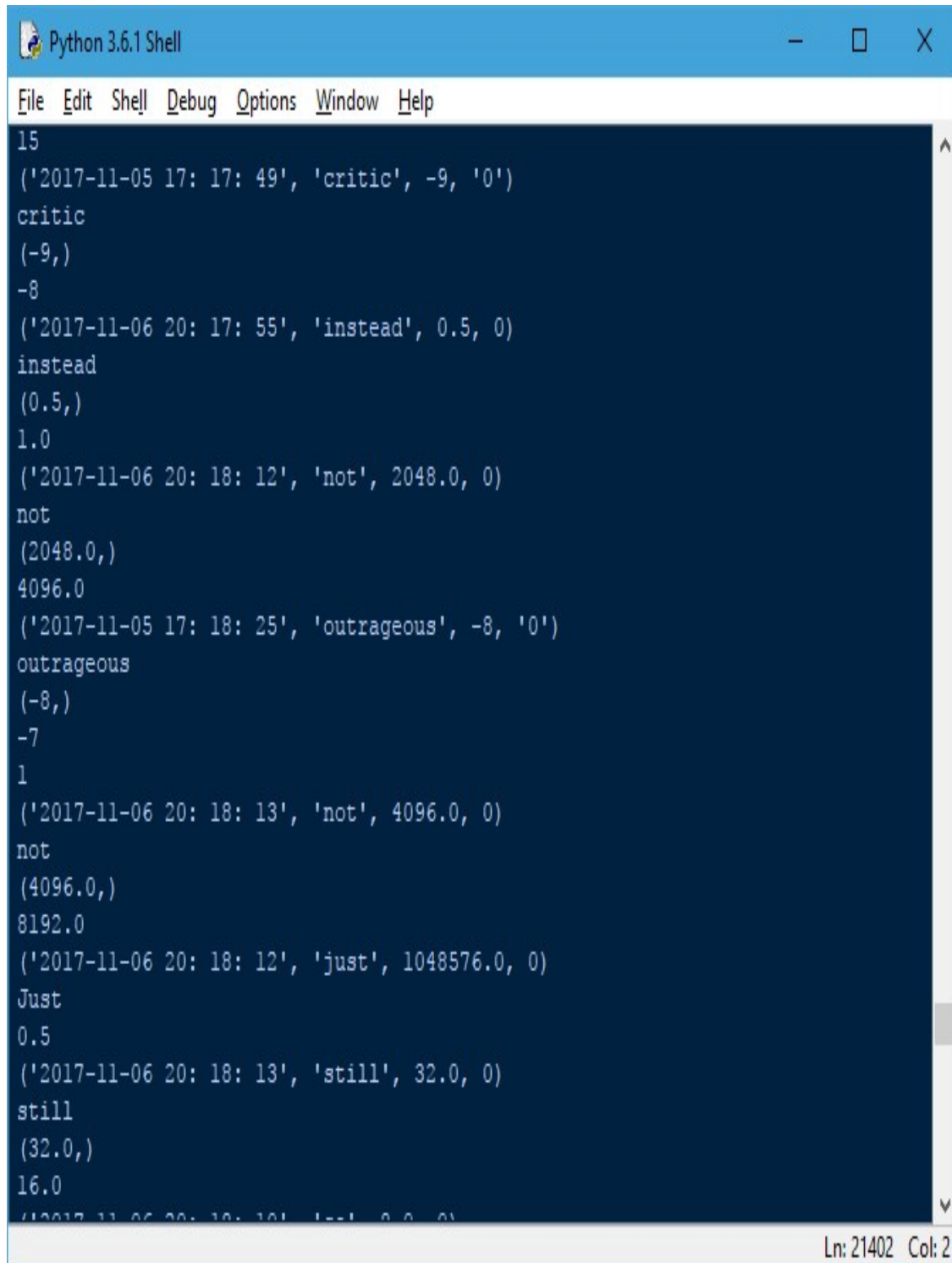
10

Fun isn't a dirty word. It's also not a word that means &#
```

Ln: 21402 Col: 2

Figure 12: Review Rating

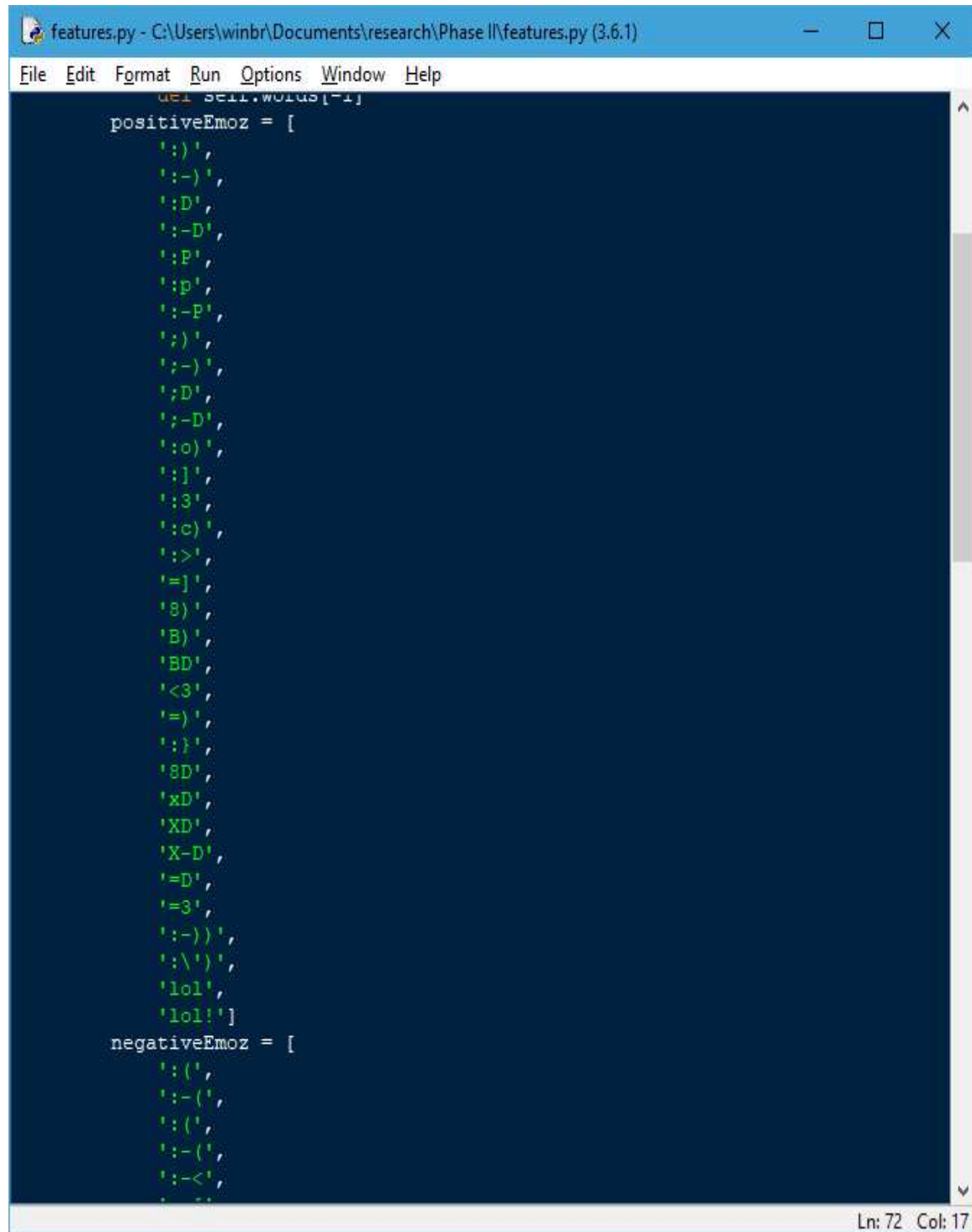
#### 5.2.2.4. UDPATING DICTIONARY

A screenshot of a Python 3.6.1 Shell window. The window has a blue title bar with the text "Python 3.6.1 Shell" and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main area is a dark blue terminal with white text. It shows a series of dictionary updates. Each update consists of a timestamp, a word, a score, and a count. The updates are: 1. ('2017-11-05 17: 17: 49', 'critic', -9, '0') with output 'critic' and '(-9,)' and line number 15. 2. ('2017-11-06 20: 17: 55', 'instead', 0.5, 0) with output 'instead' and '(0.5,)' and line number -8. 3. ('2017-11-06 20: 18: 12', 'not', 2048.0, 0) with output 'not' and '(2048.0,)' and line number 4096.0. 4. ('2017-11-05 17: 18: 25', 'outrageous', -8, '0') with output 'outrageous' and '(-8,)' and line number -7. 5. ('2017-11-06 20: 18: 13', 'not', 4096.0, 0) with output 'not' and '(4096.0,)' and line number 8192.0. 6. ('2017-11-06 20: 18: 12', 'just', 1048576.0, 0) with output 'Just' and '0.5' and line number 16.0. 7. ('2017-11-06 20: 18: 13', 'still', 32.0, 0) with output 'still' and '(32.0,)' and line number 16.0. The status bar at the bottom right shows "Ln: 21402 Col: 2".

```
Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
15
('2017-11-05 17: 17: 49', 'critic', -9, '0')
critic
(-9,)
-8
('2017-11-06 20: 17: 55', 'instead', 0.5, 0)
instead
(0.5,)
1.0
('2017-11-06 20: 18: 12', 'not', 2048.0, 0)
not
(2048.0,)
4096.0
('2017-11-05 17: 18: 25', 'outrageous', -8, '0')
outrageous
(-8,)
-7
1
('2017-11-06 20: 18: 13', 'not', 4096.0, 0)
not
(4096.0,)
8192.0
('2017-11-06 20: 18: 12', 'just', 1048576.0, 0)
Just
0.5
('2017-11-06 20: 18: 13', 'still', 32.0, 0)
still
(32.0,)
16.0
('2017-11-06 20: 18: 13', 'still', 32.0, 0)
Ln: 21402 Col: 2
```

Figure 13: Dictionary Updating

### 5.2.2.5. EMOJI DATASET

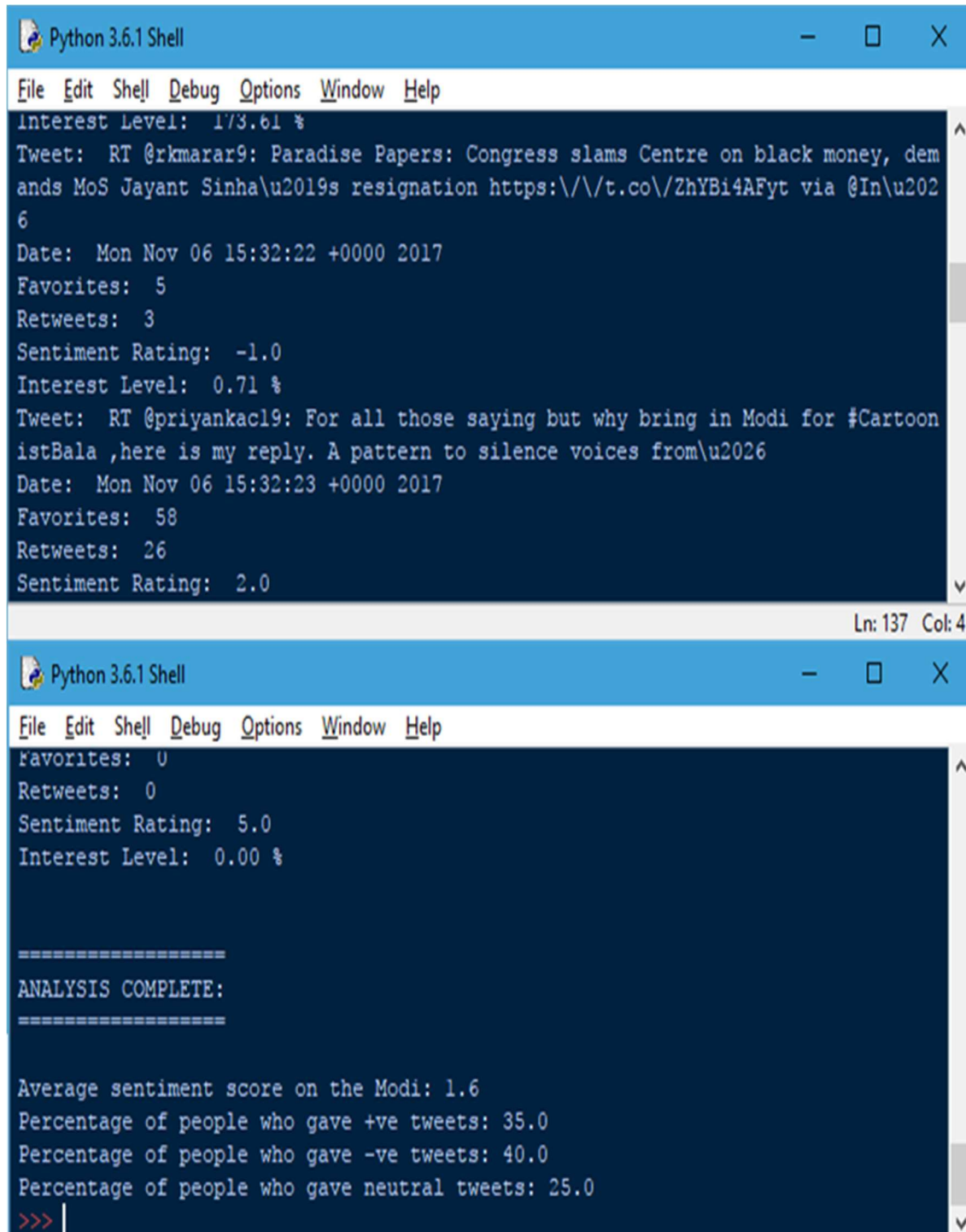
A screenshot of a Python script editor window titled 'features.py - C:\Users\winbr\Documents\research\Phase II\features.py (3.6.1)'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The script content is as follows:

```
def split_words[-1]
positiveEmoz = [
    ':)',
    ':~)',
    ':D',
    ':~D',
    ':P',
    ':p',
    ':~P',
    ':)',
    ':~)',
    ':D',
    ':~D',
    ':o)',
    ':]',
    ':3',
    ':c)',
    ':>',
    '=]',
    '8)',
    'B)',
    'BD',
    '<3',
    '=)',
    ':}',
    '8D',
    'xD',
    'XD',
    'X-D',
    '=D',
    '=3',
    ':~))',
    ':\'\'',
    'lol',
    'lol!']
negativeEmoz = [
    ':(',
    ':~(',
    ':(',
    ':~(',
    ':~<',
    ':~<'
```

The status bar at the bottom right indicates 'Ln: 72 Col: 17'.

Figure 14: Emoji Dataset

### 5.2.2.6. TWITTER ANALYSIS



```
Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
Interest Level: 1/3.61 %
Tweet: RT @rkmarar9: Paradise Papers: Congress slams Centre on black money, dem
ands MoS Jayant Sinha's resignation https://t.co/ZhYBi4AFyt via @In\202
6
Date: Mon Nov 06 15:32:22 +0000 2017
Favorites: 5
Retweets: 3
Sentiment Rating: -1.0
Interest Level: 0.71 %
Tweet: RT @priyankacl9: For all those saying but why bring in Modi for #Cartoon
istBala ,here is my reply. A pattern to silence voices from\2026
Date: Mon Nov 06 15:32:23 +0000 2017
Favorites: 58
Retweets: 26
Sentiment Rating: 2.0
Ln: 137 Col: 4

Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
Favorites: 0
Retweets: 0
Sentiment Rating: 5.0
Interest Level: 0.00 %

=====
ANALYSIS COMPLETE:
=====

Average sentiment score on the Modi: 1.6
Percentage of people who gave +ve tweets: 35.0
Percentage of people who gave -ve tweets: 40.0
Percentage of people who gave neutral tweets: 25.0
>>> |
```

Figure 15: Twitter Analysis

### 5.2.3. GRAPH ANALYSIS

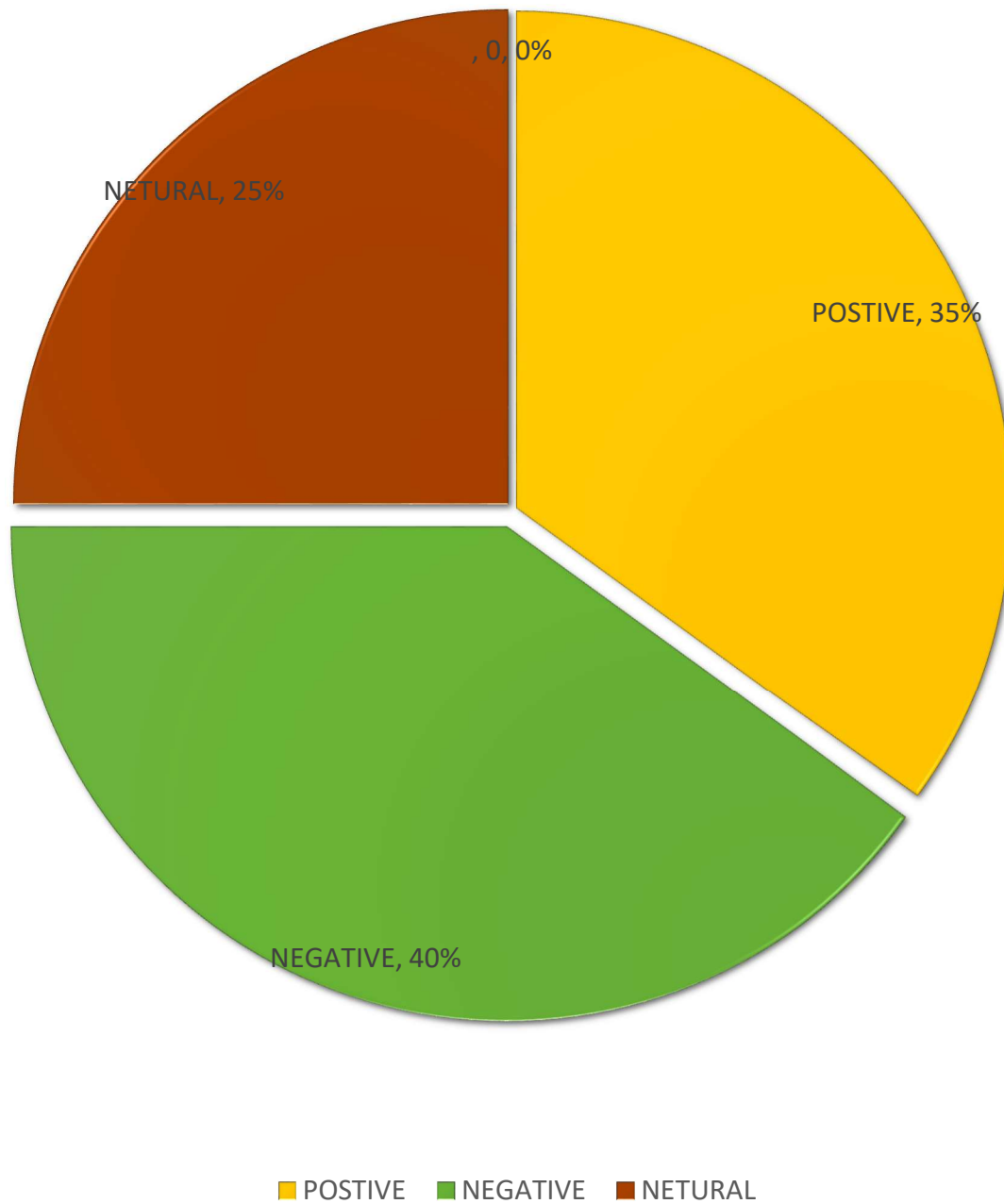


Figure 16: Phase II, Graph Analysis

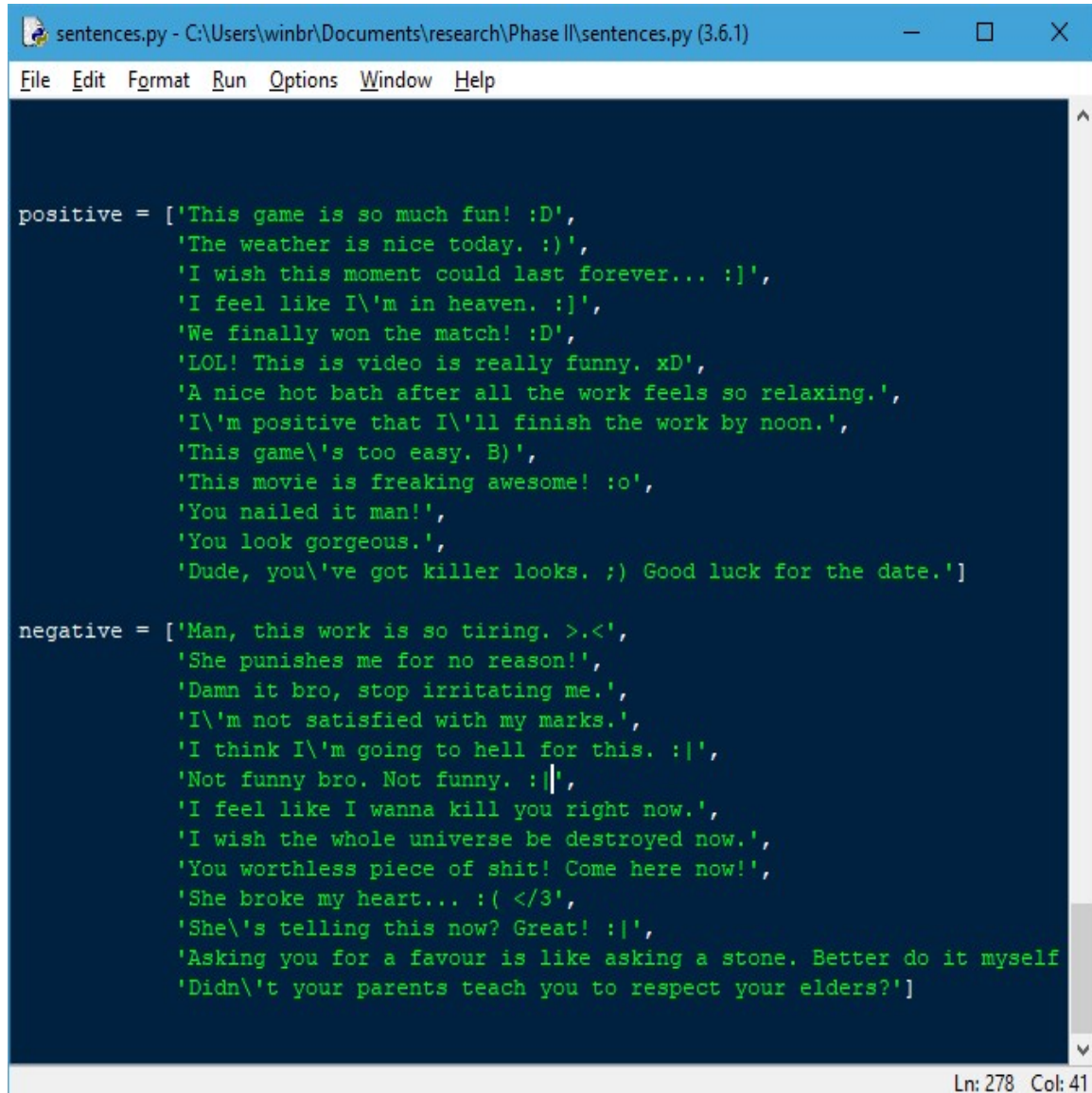
#### **5.2.4. CONCLUSION**

After examining the case study , Average sentiment score on Modi is 1.6. Percentage of people who gave positive tweets are 35.0 while Negative Percentage of Tweets are 40.0 and Percentage of people who gave neutral tweets are 25.0.



## 5.3. COMBINE PHASE 1 AND 2

### 5.3.1. DATA SETS



```
sentences.py - C:\Users\winbr\Documents\research\Phase II\sentences.py (3.6.1)
File Edit Format Run Options Window Help

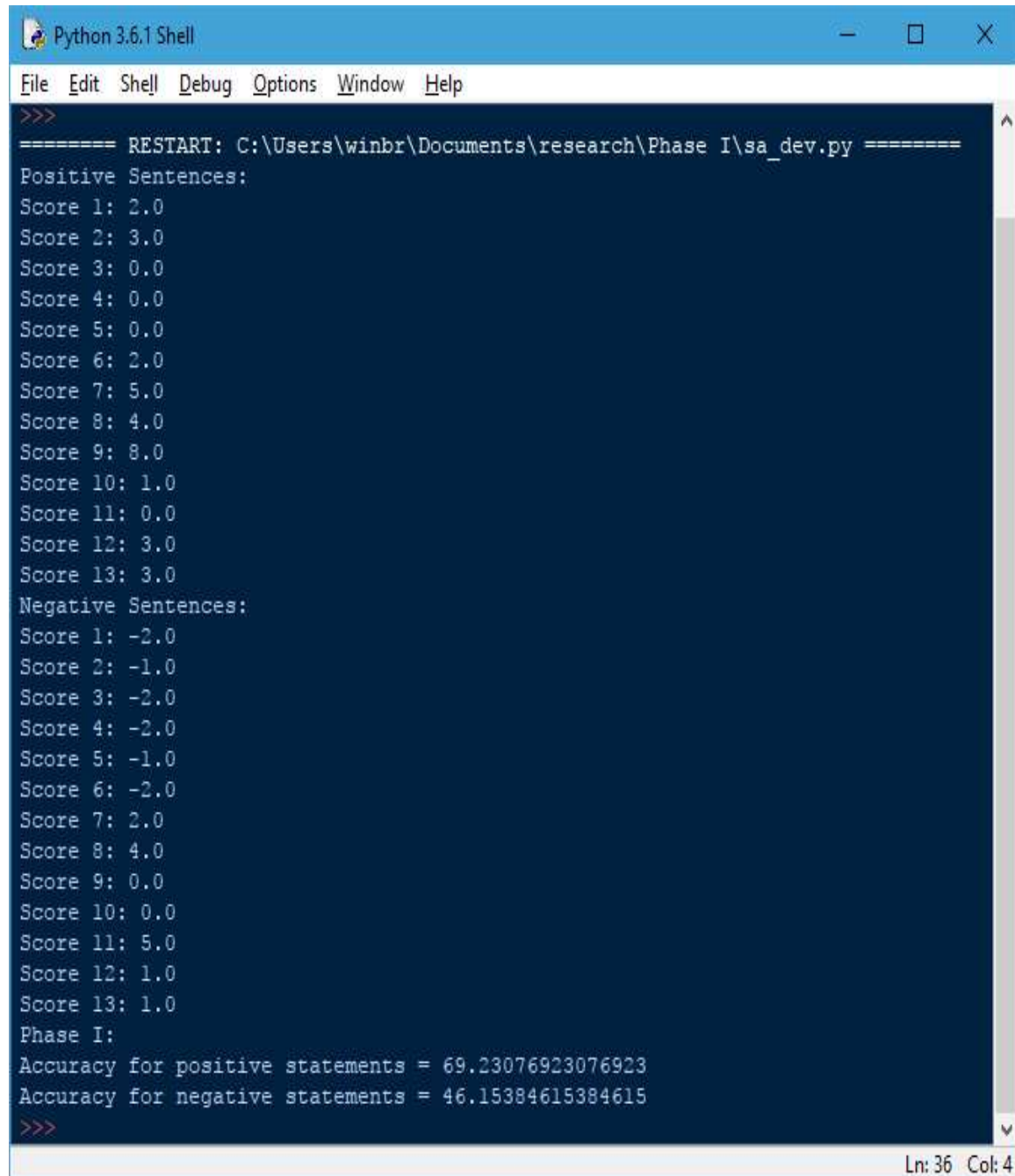
positive = ['This game is so much fun! :D',
            'The weather is nice today. :)',
            'I wish this moment could last forever... :}',
            'I feel like I\'m in heaven. :}',
            'We finally won the match! :D',
            'LOL! This is video is really funny. xD',
            'A nice hot bath after all the work feels so relaxing.',
            'I\'m positive that I\'ll finish the work by noon.',
            'This game\'s too easy. B)',
            'This movie is freaking awesome! :o',
            'You nailed it man!',
            'You look gorgeous.',
            'Dude, you\'ve got killer looks. ;) Good luck for the date.'].

negative = ['Man, this work is so tiring. >.<',
            'She punishes me for no reason!',
            'Damn it bro, stop irritating me.',
            'I\'m not satisfied with my marks.',
            'I think I\'m going to hell for this. :|',
            'Not funny bro. Not funny. :||',
            'I feel like I wanna kill you right now.',
            'I wish the whole universe be destroyed now.',
            'You worthless piece of shit! Come here now!',
            'She broke my heart... :( </3',
            'She\'s telling this now? Great! :|',
            'Asking you for a favour is like asking a stone. Better do it myself',
            'Didn\'t your parents teach you to respect your elders?']

Ln: 278 Col: 41
```

Figure 17: Test Dataset

### 5.3.2. PHASE 1 RESULT



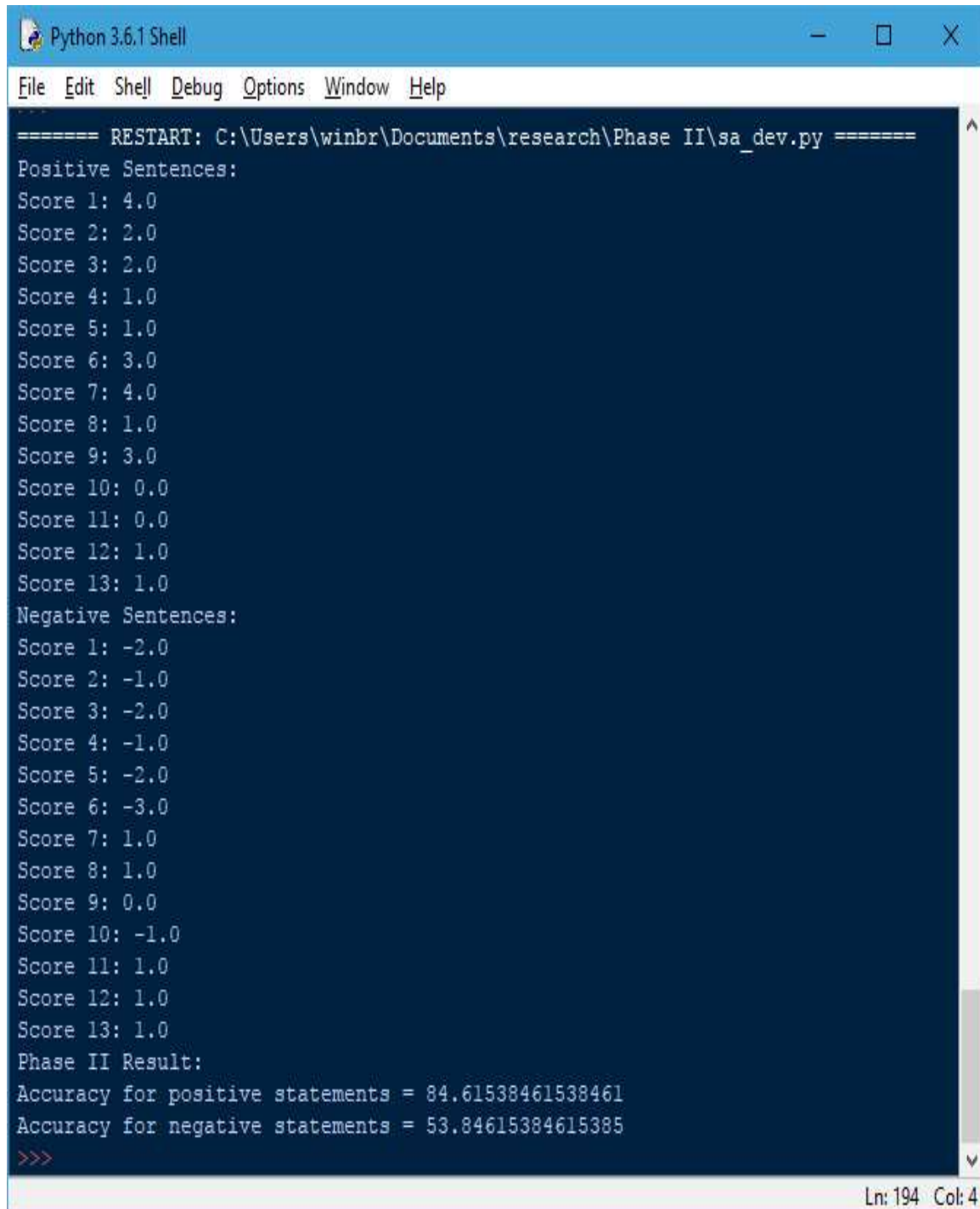
```
Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
>>>
===== RESTART: C:\Users\winbr\Documents\research\Phase I\sa_dev.py =====
Positive Sentences:
Score 1: 2.0
Score 2: 3.0
Score 3: 0.0
Score 4: 0.0
Score 5: 0.0
Score 6: 2.0
Score 7: 5.0
Score 8: 4.0
Score 9: 8.0
Score 10: 1.0
Score 11: 0.0
Score 12: 3.0
Score 13: 3.0
Negative Sentences:
Score 1: -2.0
Score 2: -1.0
Score 3: -2.0
Score 4: -2.0
Score 5: -1.0
Score 6: -2.0
Score 7: 2.0
Score 8: 4.0
Score 9: 0.0
Score 10: 0.0
Score 11: 5.0
Score 12: 1.0
Score 13: 1.0
Phase I:
Accuracy for positive statements = 69.23076923076923
Accuracy for negative statements = 46.15384615384615
>>>
```

Ln: 36 Col: 4

Figure 18: Phase I Result



### 5.3.3. PHASE 2 RESULT



```
Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:\Users\winbr\Documents\research\Phase II\sa_dev.py =====
Positive Sentences:
Score 1: 4.0
Score 2: 2.0
Score 3: 2.0
Score 4: 1.0
Score 5: 1.0
Score 6: 3.0
Score 7: 4.0
Score 8: 1.0
Score 9: 3.0
Score 10: 0.0
Score 11: 0.0
Score 12: 1.0
Score 13: 1.0
Negative Sentences:
Score 1: -2.0
Score 2: -1.0
Score 3: -2.0
Score 4: -1.0
Score 5: -2.0
Score 6: -3.0
Score 7: 1.0
Score 8: 1.0
Score 9: 0.0
Score 10: -1.0
Score 11: 1.0
Score 12: 1.0
Score 13: 1.0
Phase II Result:
Accuracy for positive statements = 84.61538461538461
Accuracy for negative statements = 53.84615384615385
>>>
```

Ln: 194 Col: 4

Figure 19: Phase II Result

### 5.3.4. GRAPH ANALYSIS

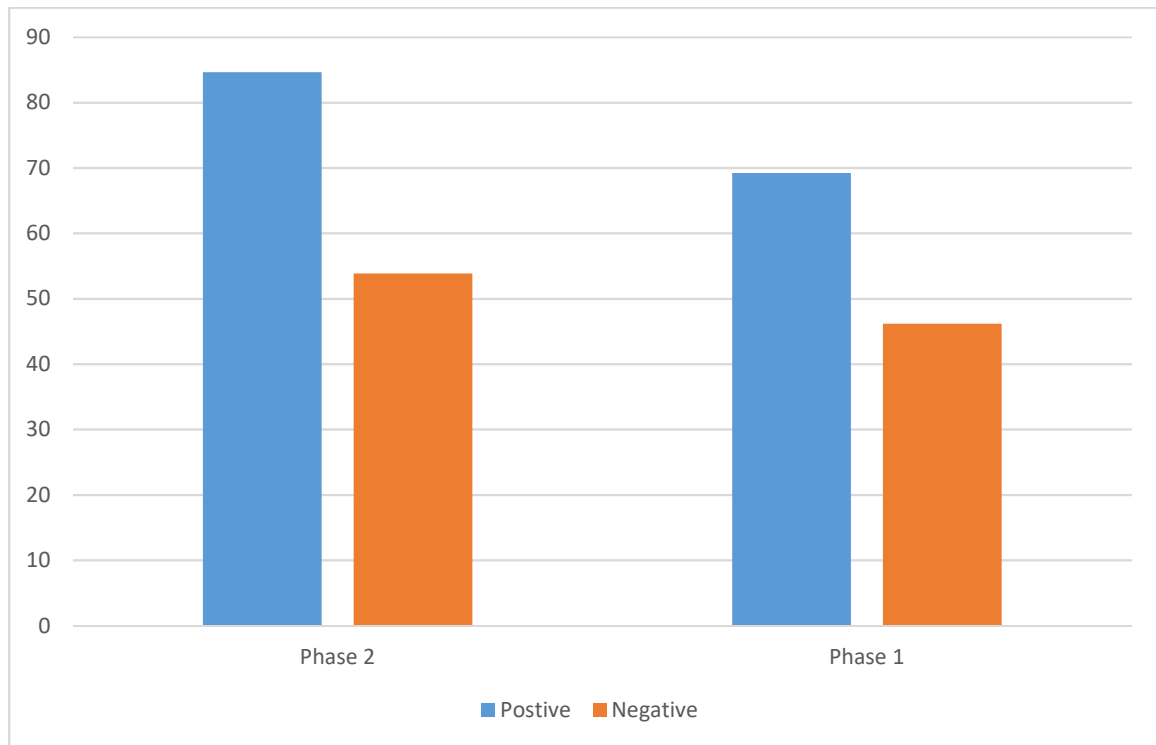


Figure 20: Phase I & II, Graph Analysis

### 5.3.5. CONCLUSION

After examining both **phase1 (without emoji detection)** and **phase 2(with emoji detection)**, we found out **the accuracy score calculated by phase 2 was better than phase 1**. Hence, we prefer using an efficient algorithm using emoji detection for sentiment analysis.

## CHAPTER-6

### CONCLUSION

#### 6.1. CONCLUSION

Sentiment analysis is an evolving field with a variety of use applications. Although sentiment analysis tasks are challenging due to their natural language processing origins, much progress has been made over the last few years due to the high demand for it.

A sentiment evaluation method using Lexicon based approach has been proposed along with detailed descriptions of each step. Experiments for both for **pre existing algorithm** and **Emoticons Detection** have been performed.

#### 6.2. DRAWBACKS ENCOUNTERED

1. We have finite number of words in lexicons/data dictionary scope is limited.
2. Sarcasm handling cannot be achieved.
3. Words defined the data dictionaries have a specific sentiment attached to it: assignation of a fixed sentiment orientation and score of the word.

## CHAPTER-7

### FUTURE WORK

1.**Sarcasm Detection** : Sarcasm is a type of sentiment where people express their negative feelings using positive or intensified positive words in the text. Sarcasm requires some shared knowledge between speaker and audience; it is a profoundly contextual phenomenon. Most computational approaches to sarcasm detection, however, treat it as a purely linguistic matter, using information such as lexical cues and their corresponding sentiment as predictive features. We show that by including extra-linguistic information from the context of an utterance on Twitter – such as properties of the author, the audience and the immediate communicative environment – we are able to achieve gains in accuracy compared to purely linguistic features in the detection of this complex phenomenon, while also shedding light on features of interpersonal interaction that enable sarcasm in conversation

2.Build an **efficient algorithm or refine the pre existing algorithm** (for sentimental analysis including sarcasm and emoticons detection) and **compare results** .

## REFERENCES

1. <https://arxiv.org/ftp/arxiv/papers/1509/1509.04219.pdf>
2. <https://www.brandwatch.com/blog/understanding-sentiment-analysis/>
3. <https://www.cs.uic.edu/~liub/publications/emnlp2016.pdf>
4. <https://www.cs.uic.edu/~liub/publications/emnlp2017.pdf>
5. <https://globallogic.com/wp-content/uploads/2014/10/Introduction-to-Sentiment-Analysis.pdf>
6. <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-015-0015-2>
7. <https://www.safaribooksonline.com/library/view/sentiment-analysis-in/9780128044384/B9780128044124000164.xhtml>
8. <https://security-informatics.springeropen.com/articles/10.1186/s13388-015-0024->

[x](#)