A MINI PROJECT REPORT

**On**

**Sentimental Analysis of YouTube Videos**

**Submitted by**

**Arya Gupta (161500131)**

**Hritik Goyal (161500245)**

**Kajal Gupta (161500252)**

**Kamal Varshney (161500253)**

**Sahil Vashishtha (161500474)**

**Supervised by**

**Mr. Vaibhav Diwan**

(Technical Trainer)

Department of Computer Engineering & Applications

**Institute of Engineering & Technology**



**GLA University**

**Mathura- 281406, INDIA**

**January, 2019**

**Department of Computer Engineering and Applications**

**GLA University, Mathura**

**17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,**

**Mathura – 281406**



**Declaration**

We hereby declare that the work which is being presented in the Mini Project “**Sentimental Analysis of YouTube Videos”,** in partial fulfillment of the requirements for Mini-Project LAB, is an authentic record of our own work carried under the supervision of **Mr. Vaibhav Diwan, Technical Trainer, GLA University, Mathura**.

**Arya Gupta**

**Hritik Goyal**

**Kajal Gupta**

**Kamal Varshney**

**Sahil Vashishtha**



**Department of Computer Engineering and Applications**

**GLA University, Mathura**

**17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,**

**Mathura – 281406**

**CERTIFICATE**

This is to certify that the project entitled **“Sentimental Analysis of YouTube Videos”** carried out in Mini Project – II Lab is a bonafide work done by Arya Gupta (16500131), **Hritik Goyal (161500244), Kajal Gupta (161500252), Kamal Varshney (161500253) and Sahil Vashishtha (161500474)** and is submitted in partial fulfillment of the requirements for the award of the degree Bachelor of Technology (Computer Science & Engineering).

**Signature of Supervisor:**

**Name of Supervisor:**

**Date:**

**ACKNOWLEDGEMENT**

It gives us a great sense of pleasure to present the report of the B. Tech Mini Project undertaken during B. Tech. Third Year. This project in itself is an acknowledgement to the inspiration, drive and technical assistance contributed to it by many individuals. This project would never have seen the light of the day without the help and guidance that we have received

Our heartiest thanks to **Dr. (Prof). Anand Singh Jalal,** Head of Dept., Department of CEA for providing us with an encouraging platform to develop this project, which thus helped us in shaping our abilities towards a constructive goal.

We owe special debt of gratitude to **Mr. Vaibhav Diwan,** Technical Trainer Department of CEA, for his constant support and guidance throughout the course of our work. His sincerity, thoroughness and perseverance have been a constant source of inspiration for us. He has showered us with all his extensively experienced ideas and insightful comments at virtually all stages of the project & has also taught us about the latest industry-oriented technologies.

We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind guidance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contribution in the completion of the project.

Arya Gupta

Hritik Goyal

Kajal Gupta

Kamal Varshney

Sahil Vashishtha

**Abstract**

Extracting speaker sentiment from natural audio streams such as YouTube is challenging. A number of factors contribute to the task difficulty, namely, Automatic Speech Recognition (ASR) of spontaneous speech, unknown background environments, variable source and channel characteristics, accents, diverse topics, *etc*.

Sentiment analysis or opinion mining is the field of study related to analyze opinions,

sentiments, evaluations, attitudes, and emotions of users which they express on social media and other online resources. The revolution of social media sites has also attracted the users towards video sharing sites, such as YouTube. The online users express their opinions or sentiments on the videos that they watch on such sites. This paper presents a brief survey of techniques to analyze opinions posted by users about a particular video.

**Table of Contents**

|  |  |
| --- | --- |
| Declaration ii |  |
| Certificate iii |  |
| Acknowledgments iv |  |
| Abstract v |  |
| Table of Contents vi |  |
| 1. **Introduction 1** |  |
| * 1. Overview………………………………………………… 1 |  |
| * 1. Motivation ………………………………………………. 2 2 ………………………..………………………………………………............. |  |
|  |  |
| 1. **Software Requirement Analysis 3** |  |
| 2.1 Tools Used………….……………………………………. 3 …………………………………………………….. |  |
| 2.2 Problem ………………………………………………….. 3  2.3 Define the module and their functionalities……….……... 4  2.4 Requirements……………………………………………… 10  ………..………………………………… |  |
| 1. **Software Design 11** |  |
| * 1. Sequence Diagram……………………………………….. 11 |  |
| 3.2 Data Flow Diagram………………………………………. 12 |  |
|  |  |
| 1. **Libraries** …………………………………………………….. **13** |  |
| 1. **Sample Code** ……………………………………………….. **38** 2. **References/Bibliography**……………………………………. **43** |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Chapter - 1** **Introduction**

* 1. **Overview**

Social networking applications such as Twitter, Facebook, YouTube, etc. are popularly used to express one's sentiment and/or opinion on a variety of topics. A large number of these applications rely on text as the main medium of communication. However, websites such as YouTube use video/audio as the primary source of communicating information. For example, "unboxing" is a very popular theme on YouTube where users express their opinion and sentiment about products while unpacking and experiencing the product for the first time. Sentiment systems that can crawl and mine these information resources can assist in establishing the popular sentiment or the "word of mouth" on a large range of topics. Such information can be tremendously useful to businesses and consumers alike.

Text-based sentiment analysis has been well researched and numerous techniques that mine reviews for opinions have been developed. However, audio-based sentiment analysis remains under explored. Recently, we had shown that audio sentiment extraction with good accuracy is possible using a combination of NLP (natural language processing) and ASR (automatic speech recognition) techniques. Particularly, we had demonstrated the capability of automatically predicting the polarity of sentiment (positive or negative). First, audio was extracted from the YouTube video and then converted to text using the ASR system, and finally the text-based sentiment system predicted the sentiment polarity. The text-based sentiment system used parts-of-speech tagging technique to automatically extract text-features, which were then employed in a maximum entropy based classification system to predict sentiment polarity.

**1.2 Motivation**

When we do a YouTube search for videos, more often than not, it happens that the caption displayed at the bottom of video player doesn’t match appropriately with the content displayed. Our project aims to analyze the caption displayed and match it with the video content for the appropriateness, & in case of faulty or wrong caption, suggest a correct caption sequence.

It also aims to perform sentiment analysis on YouTube videos & perform categorization.

**Chapter – 2**

**Software Requirement Analysis**

**2.1 Tools Used**

**Python IDLE -** is an [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) for [Python](https://en.wikipedia.org/wiki/Python_(programming_language)), which has been bundled with the default implementation of the language. IDLE is intended to be a simple [IDE](https://en.wikipedia.org/wiki/Integrated_development_environment) and suitable for beginners, especially in an educational environment. To that end, it is cross-platform, and avoids feature clutter.

**Natural Language Toolkit -** NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.

NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

[Natural Language Processing with Python](http://nltk.org/book) provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The online version of the book has been updated for Python 3 and NLTK 3.

**2.2 Problem**

YouTube is overall a great platform, which explains how it has been popular for so long. However it does have its own share of flaws. Two of the biggest problems most users face are bad recommendation videos, and spam comments. It could help in better subtitles generation.

**2.3 Modules and their Functionality**

**1. Interface of the application**

In this module we use Tkinter:

Tk/Tcl has long been an integral part of Python. It provides a robust and platform independent windowing toolkit, that is available to Python programmers using the tkinter package, and its extension, the [tkinter.tix](https://docs.python.org/3/library/tkinter.tix.html#module-tkinter.tix) and the [tkinter.ttk](https://docs.python.org/3/library/tkinter.ttk.html#module-tkinter.ttk) modules.

The [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) package is a thin object-oriented layer on top of Tcl/Tk. To use [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter), you don’t need to write Tcl code, but you will need to consult the Tk documentation, and occasionally the Tcl documentation. [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) is a set of wrappers that implement the Tk widgets as Python classes. In addition, the internal module \_tkinterprovides a threadsafe mechanism which allows Python and Tcl to interact.

[tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter)’s chief virtues are that it is fast, and that it usually comes bundled with Python. Although its standard documentation is weak, good material is available, which includes: references, tutorials, a book and others. [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) is also famous for having an outdated look and feel, which has been vastly improved in Tk 8.5.

**2. Video To Audio**

In this module we use MoviePy Api.

**MoviePy** : MoviePy is an open source software originally written by Zulko and released under the MIT licence. It works on Windows, Mac, and Linux, with Python 2 or Python 3. The code is hosted on Github, where you can push improvements, report bugs and ask for help. There is also a MoviePy forum on Reddit and a mailing list on librelist**.**

MoviePy is a Python module for video editing, which can be used for basic operations (like cuts, concatenations, title insertions), video compositing (a.k.a. non-linear editing), video processing, or to create advanced effects. It can read and write the most common video formats, including GIF.

**Do I need MoviePy ?**

Here are a few reasons why you may want to edit videos in Python:

* You have many videos to process or to compose in a complicated way.
* You want to automatize the creation of videos or GIFs on a web server (Django, Flask, etc.)
* You want to automatize tedious tasks, like title insertions tracking objects, cuting scenes, making end credits, subtitles, etc...
* You want to code your own video effects to do something no existing video editor can.
* You want to create animations from images generated by another python library (Matplotlib, Mayavi, Gizeh, scikit-images...)

And here are a few uses for which MoviePy is NOT the best solution:

* You only need to do frame-by-frame video analysis (with face detection or other fancy stuff). This could be done with MoviePy in association with other libraries,
* You only want to convert a video file, or turn a series of image files into a movie. In this case it is better to directly call ffmpeg (or avconv or mencoder...) it will be faster more memory-efficient than going through MoviePy.

**Advantages and limitations**

MoviePy has been developed with the following goals in mind:

* Simple an intuitive. Basic operations can be done in one line. The code is easy to learn and easy to understand for newcomers.
* Flexible. You have total control over the frames of the video and audio, and creating your own effects is easy as Py.
* Portable. The code uses very common software (Numpy and FFMPEG) and can run on (almost) any machine with (almost) any version of Python.

For the limitations: MoviePy cannot (yet) stream videos (read from a webcam, or render a video live on a distant machine), and is not really designed for video processing involving many successive frames of a movie (like video stabilization, you’ll need another software for that). You can also have memory problems if you use many video, audio, and image sources at the same time (>100), but this will be fixed in future versions.

**3. Audio To Text**

In this module we use **Speech recognition Api.**

Speech recognition is the process of converting spoken words to text. Python supports many speech recognition engines and APIs, including Google Speech Engine, Google Cloud Speech API, Microsoft Bing Voice Recognition and IBM Speech to Text.

The first component of speech recognition is, of course, speech. Speech must be converted from physical sound to an electrical signal with a microphone, and then to digital data with an analog-to-digital converter. Once digitized, several models can be used to transcribe the audio to text.

Most modern speech recognition systems rely on what is known as a [Hidden Markov Model](https://en.wikipedia.org/wiki/Hidden_Markov_model)(HMM). This approach works on the assumption that a speech signal, when viewed on a short enough timescale (say, ten milliseconds), can be reasonably approximated as a stationary process—that is, a process in which statistical properties do not change over time.

**Speech-to-Text API recognition**

A Speech-to-Text API synchronous recognition request is the simplest method for performing recognition on speech audio data. Speech-to-Text can process up to 1 minute of speech audio data sent in a synchronous request. After Speech-to-Text processes and recognizes all of the audio, it returns a response.

A synchronous request is blocking. Speech-to-Text typically processes audio faster than realtime, processing 30 seconds of audio in 15 seconds on average. In cases of poor audio quality, your recognition request can take significantly longer.

All Speech-to-Text API synchronous recognition requests must include a speech recognition config field (of type RecognitionConfig). A RecognitionConfig contains the following sub-fields:

**encoding** - (required) specifies the encoding scheme of the supplied audio (of type AudioEncoding). If you have a choice in codec, prefer a lossless encoding such as FLAC or LINEAR16 for best performance. (For more information, see Audio Encodings.) The encoding field is optional for FLAC and WAV files where the encoding is included in the file header.

**sampleRateHertz** - (required) specifies the sample rate (in Hertz) of the supplied audio. (For more information on sample rates, see Sample Rates below.) The sampleRateHertz field is optional for FLAC and WAV files where the sample rate is included in the file header.

**languageCode** - (required) contains the language + region/locale to use for speech recognition of the supplied audio. The language code must be a BCP-47 identifier. Note that language codes typically consist of primary language tags and secondary region subtags to indicate dialects (for example, 'en' for English and 'US' for the United States in the above example.) (For a list of supported languages, see Supported Languages.)

**maxAlternatives** - (optional, defaults to 1) indicates the number of alternative transcriptions to provide in the response. By default, the Speech-to-Text API provides one primary transcription. If you wish to evaluate different alternatives, set maxAlternatives to a higher value. Note that Speech-to-Text will only return alternatives if the recognizer determines alternatives to be of sufficient quality; in general, alternatives are more appropriate for real-time requests requiring user feedback (for example, voice commands) and therefore are more suited for streaming recognition requests.

**profanityFilter** - (optional) indicates whether to filter out profane words or phrases. Words filtered out will contain their first letter and asterisks for the remaining characters (e.g. f\*\*\*). The profanity filter operates on single words, it does not detect abusive or offensive speech that is a phrase or a combination of words.

**speechContext** - (optional) contains additional contextual information for processing this audio. A context contains the following sub-field:

**phrases** - contains a list of words and phrases that provide hints to the speech recognition task.

* Audio is supplied to Speech-to-Text through the audio parameter of type RecognitionAudio. The audio field contains either of the following sub-fields:
* Content contains the audio to evaluate, embedded within the request. See Embedding Audio Content below for more information. Audio passed directly within this field is limited to 1 minute in duration.
* URI contains a URI pointing to the audio content. The file must not be compressed (for example, gzip). Currently, this field must contain a Google Cloud Storage URI (of format gs://bucket-name/path\_to\_audio\_file). See Passing Audio reference by a URI below.)

**Languages** -Speech-to-Text's recognition engine supports a variety of languages and dialects. You specify the language (and national or regional dialect) of your audio within the request configuration's languageCode field, using a BCP-47 identifier.

A full list of supported languages, and explanation of BCP-47 identifier tags, is available on the Language Support page.

**4. Text To Polarity Analysis**

In this module we use **VADER Sentiment Analysis.**

**VADER** (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media. VADER uses a combination of A sentiment lexicon is a list of lexical features (e.g., words) which are generally labelled according to their semantic orientation as either positive or negative.

VADER has been found to be quite successful when dealing with social media texts, NY Times editorials, movie reviews, and product reviews. This is because VADER not only tells about the Positivity and Negativity score but also tells us about how positive or negative a sentiment is.

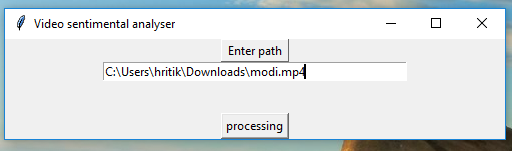
It is fully open-sourced under the MIT License. The developers of VADER have used Amazon’s Mechanical Turk to get most of their ratings, You can find complete details on their Github Page.

**Advantages of using VADER**

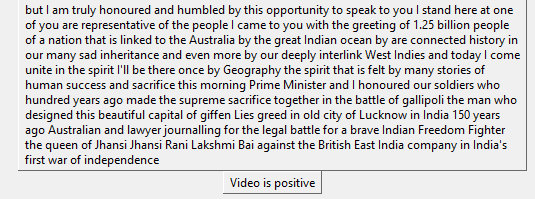
VADER has a lot of advantages over traditional methods of Sentiment Analysis, including:

* It works exceedingly well on social media type text, yet readily generalizes to multiple domains
* It doesn’t require any training data but is constructed from a generalizable, valence-based, human-curated gold standard sentiment lexicon
* It is fast enough to be used online with streaming data, and
* It does not severely suffer from a speed-performance tradeoff.

**Input Format**

****

**Output Format**

****

**2.4 Requirements**

1. **Hardware Requirements**

* Minimum 4 GB RAM, and
* Minimum i3 processor.

1. **Software Requirements**

* Any windows based operating system,
* Python,
* NLTK, and
* NLP Libraries.

**Chapter – 3**

**Software Design**

**3.1 Sequence Diagram**

Sequence Diagram is an interaction diagram that details how operations are carried out -- what messages are sent and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence.

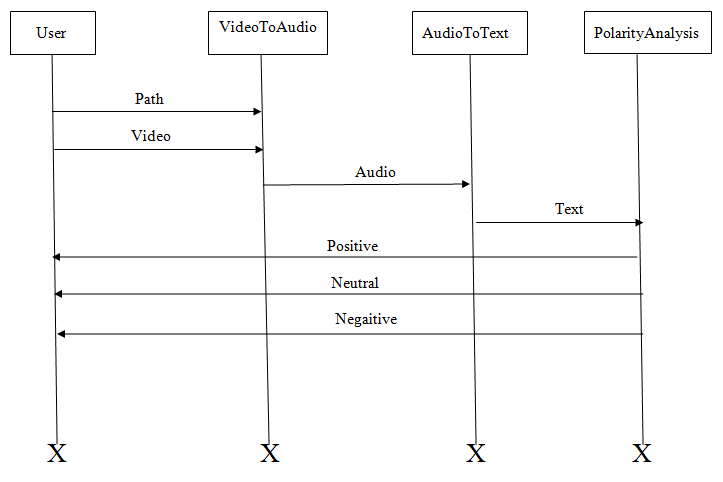


Fig. 3.1 Sequence Diagram

**3.2 Data Flow Diagram**

A Data Flow Diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range from simple even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled. They can be used to analyze an existing system or model a new one. Like all the best diagrams and charts, a DFD can often visually “say” things that would be hard to explain in words, and they work for both technical and non technical audiences, from developer to CEO. That’s why DFDs is remains so popular after all these years. While they work well for dataflow software and systems, they are less applicable nowadays to visualizing interactive, real-time or database-oriented software or systems.

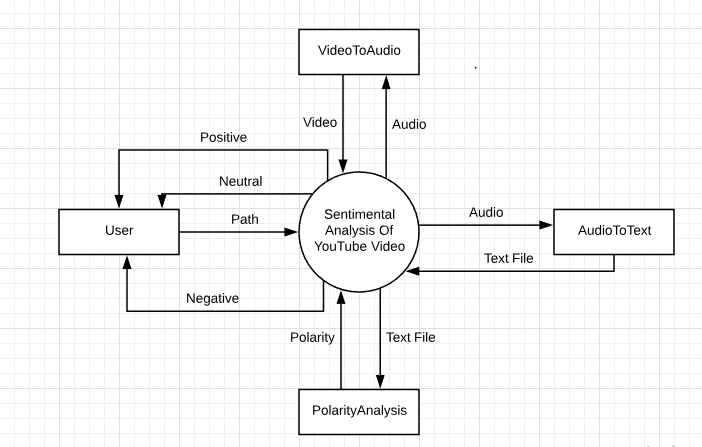


Fig. 3.2.1 Level 0 DFD

**Chapter – 4**

**Libraries**

**4.1 Tkinter**:

Tk/Tcl has long been an integral part of Python. It provides a robust and platform independent windowing toolkit, that is available to Python programmers using the [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) package, and its extension, the [tkinter.tix](https://docs.python.org/3/library/tkinter.tix.html#module-tkinter.tix) and the [tkinter.ttk](https://docs.python.org/3/library/tkinter.ttk.html#module-tkinter.ttk) modules.

The [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) package is a thin object-oriented layer on top of Tcl/Tk. To use [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter), you don’t need to write Tcl code, but you will need to consult the Tk documentation, and occasionally the Tcl documentation. [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) is a set of wrappers that implement the Tk widgets as Python classes. In addition, the internal module \_tkinterprovides a threadsafe mechanism which allows Python and Tcl to interact.

[tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter)’s chief virtues are that it is fast, and that it usually comes bundled with Python. Although its standard documentation is weak, good material is available, which includes: references, tutorials, a book and others. [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) is also famous for having an outdated look and feel, which has been vastly improved in Tk 8.5.

**4.1.1 Tkinter package:**

The [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) package (“Tk interface”) is the standard Python interface to the Tk GUI toolkit. Both Tk and [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) are available on most Unix platforms, as well as on Windows systems. (Tk itself is not part of Python; it is maintained at ActiveState.)

Running python–m tkinter from the command line should open a window demonstrating a simple Tk interface, letting you know that [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) is properly installed on your system, and also showing what version of Tcl/Tk is installed, so you can read the Tcl/Tk documentation specific to that version.

**4.1.2. Tkinter Modules:**

Most of the time, [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) is all you really need, but a number of additional modules are available as well. The Tk interface is located in a binary module named \_tkinter. This module contains the low-level interface to Tk, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter.

In addition to the Tk interface module, [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) includes a number of Python modules, tkinter.constantsbeing one of the most important. Importing [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) will automatically import tkinter.constants, so, usually, to use Tkinter all you need is a simple import statement: import tkinter

# 4.1.3 Extension widgets for Tk-tkinter.tix

The [tkinter.tix](https://docs.python.org/3/library/tkinter.tix.html#module-tkinter.tix) (Tk Interface Extension) module provides an additional rich set of widgets. Although the standard Tk library has many useful widgets, they are far from complete. The [tkinter.tix](https://docs.python.org/3/library/tkinter.tix.html#module-tkinter.tix) library provides most of the commonly needed widgets that are missing from standard Tk: [HList](https://docs.python.org/3/library/tkinter.tix.html" \l "tkinter.tix.HList" \o "tkinter.tix.HList), [ComboBox](https://docs.python.org/3/library/tkinter.tix.html" \l "tkinter.tix.ComboBox" \o "tkinter.tix.ComboBox), [Control](https://docs.python.org/3/library/tkinter.tix.html#tkinter.tix.Control) (a.k.a. SpinBox) and an assortment of scrollable widgets. [tkinter.tix](https://docs.python.org/3/library/tkinter.tix.html#module-tkinter.tix) also includes many more widgets that are generally useful in a wide range of applications: [NoteBook](https://docs.python.org/3/library/tkinter.tix.html" \l "tkinter.tix.NoteBook" \o "tkinter.tix.NoteBook), [FileEntry](https://docs.python.org/3/library/tkinter.tix.html" \l "tkinter.tix.FileEntry" \o "tkinter.tix.FileEntry), [PanedWindow](https://docs.python.org/3/library/tkinter.tix.html" \l "tkinter.tix.PanedWindow" \o "tkinter.tix.PanedWindow), etc; there are more than 40 of them.

With all these new widgets, you can introduce new interaction techniques into applications, creating more useful and more intuitive user interfaces. You can design your application by choosing the most appropriate widgets to match the special needs of your application and users.

# 4.1.4. Tk themed widgets-tkinter.ttk

The [tkinter.ttk](https://docs.python.org/3/library/tkinter.ttk.html#module-tkinter.ttk) module provides access to the Tk themed widget set, introduced in Tk 8.5. If Python has not been compiled against Tk 8.5, this module can still be accessed if Tile has been installed. The former method using Tk 8.5 provides additional benefits including anti-aliased font rendering under X11 and window transparency (requiring a composition window manager on X11).

The basic idea for [tkinter.ttk](https://docs.python.org/3/library/tkinter.ttk.html#module-tkinter.ttk) is to separate, to the extent possible, the code implementing a widget’s behavior from the code implementing its appearance.

**4.1.5. Others**

* [PyGTK](http://www.pygtk.org/): PyGTK provides bindings for an older version of the library, GTK+ 2. It provides an object oriented interface that is slightly higher level than the C one. There are also bindings to [GNOME](https://www.gnome.org/). An online [tutorial](http://www.pygtk.org/pygtk2tutorial/index.html)is available.
* [PyQt](https://riverbankcomputing.com/software/pyqt/intro): PyQt is a **sip**-wrapped binding to the Qt toolkit. Qt is an extensive C++ GUI application development framework that is available for Unix, Windows and Mac OS X. **sip** is a tool for generating bindings for C++ libraries as Python classes, and is specifically designed for Python.
* [PySide](https://wiki.qt.io/PySide): PySide is a newer binding to the Qt toolkit, provided by Nokia. Compared to PyQt, its licensing scheme is friendlier to non-open source applications.
* [wxPython](https://www.wxpython.org/): wxPython is a cross-platform GUI toolkit for Python that is built around the popular [wxWidgets](https://www.wxwidgets.org/) (formerly wxWindows) C++ toolkit. It provides a native look and feel for applications on Windows, Mac OS X, and Unix systems by using each platform’s native widgets where ever possible, (GTK+ on Unix-like systems). In addition to an extensive set of widgets, wxPython provides classes for online documentation and context sensitive help, printing, HTML viewing, low-level device context drawing, drag and drop, system clipboard access, an XML-based resource format and more, including an ever growing library of user-contributed modules.

PyGTK, PyQt, and wxPython, all have a modern look and feel and more widgets than Tkinter. In addition, there are many other GUI toolkits for Python, both cross-platform, and platform-specific. See the [GUI Programming](https://wiki.python.org/moin/GuiProgramming)page in the Python Wiki for a much more complete list, and also for links to documents where the different GUI toolkits are compared.

# 4.1.6. Python interface to Tcl/Tk:

The [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) package (“Tk interface”) is the standard Python interface to the Tk GUI toolkit. Both Tk and [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) are available on most Unix platforms, as well as on Windows systems. (Tk itself is not part of Python; it is maintained at ActiveState.)

Running python–m tkinter  from the command line should open a window demonstrating a simple Tk interface, letting you know that [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) is properly installed on your system, and also showing what version of Tcl/Tk is installed, so you can read the Tcl/Tk documentation specific to that version.

Python Tkinter Resurces:

The Python Tkinker Topics Guide provides a great deal of information on using TK from Python and links to other sources of information on TK.

## 4.1.7. Tkinter Modules

Most of the time, [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) is all you really need, but a number of additional modules are available as well. The Tk interface is located in a binary module named \_tkinter. This module contains the low-level interface to Tk, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter.

In addition to the Tk interface module, [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) includes a number of Python modules, tkinter.constantsbeing one of the most important. Importing [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) will automatically import tkinter.constants, so, usually, to use Tkinter all you need is a simple import statement:

* import tkinter

Or, more often:

* from tkinter import \*

class tkinter.**Tk**(screenName=None, baseName=None, className='Tk', useTk=1)

The [Tk](https://docs.python.org/3/library/tkinter.html" \l "tkinter.Tk" \o "tkinter.Tk) class is instantiated without arguments. This creates a toplevel widget of Tk which usually is the main window of an application. Each instance has its own associated Tcl interpreter.

tkinter.**Tcl**(screenName=None, baseName=None, className='Tk', useTk=0)

The [Tcl()](https://docs.python.org/3/library/tkinter.html" \l "tkinter.Tcl" \o "tkinter.Tcl) function is a factory function which creates an object much like that created by the [Tk](https://docs.python.org/3/library/tkinter.html" \l "tkinter.Tk" \o "tkinter.Tk) class, except that it does not initialize the Tk subsystem. This is most often useful when driving the Tcl interpreter in an environment where one doesn’t want to create extraneous toplevel windows, or where one cannot (such as Unix/Linux systems without an X server). An object created by the [Tcl()](https://docs.python.org/3/library/tkinter.html" \l "tkinter.Tcl" \o "tkinter.Tcl) object can have a Toplevel window created (and the Tk subsystem initialized) by calling its loadtk() method.

* Other modules that provide Tk support include:

1. [tkinter.scrolledtext](https://docs.python.org/3/library/tkinter.scrolledtext.html#module-tkinter.scrolledtext): Text widget with a vertical scroll bar built in.

2. tkinter.colorchooser: Dialog to let the user choose a color.

3. tkinter.commondialog: Base class for the dialogs defined in the other modules listed here.

4. tkinter.filedialog: Common dialogs to allow the user to specify a file to open or save.

5. tkinter.font: Utilities to help work with fonts.

6. tkinter.messagebox: Access to standard Tk dialog boxes.

7. tkinter.simpledialog: Basic dialogs and convenience functions.

8. tkinter.dnd: Drag-and-drop support for [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter). This is experimental and should become deprecated when it is replaced with the Tk DND.

9. [turtle](https://docs.python.org/3/library/turtle.html#module-turtle): Turtle graphics in a Tk window.

## 4.1.8. Tkinter Life Preserver

This section is not designed to be an exhaustive tutorial on either Tk or Tkinter. Rather, it is intended as a stop gap, providing some introductory orientation on the system.

Credits:

* Tk was written by John Ousterhout while at Berkeley.
* Tkinter was written by Steen Lumholt and Guido van Rossum.
* This Life Preserver was written by Matt Conway at the University of Virginia.
* The HTML rendering, and some liberal editing, was produced from a FrameMaker version by Ken Manheimer.
* Fredrik Lundh elaborated and revised the class interface descriptions, to get them current with Tk 4.2.
* Mike Clarkson converted the documentation to LaTeX, and compiled the User Interface chapter of the reference manual.

### How To Use This Section

This section is designed in two parts: the first half (roughly) covers background material, while the second half can be taken to the keyboard as a handy reference.

When trying to answer questions of the form “how do I do blah”, it is often best to find out how to do “blah” in straight Tk, and then convert this back into the corresponding [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) call. Python programmers can often guess at the correct Python command by looking at the Tk documentation. This means that in order to use Tkinter, you will have to know a little bit about Tk. This document can’t fulfill that role, so the best we can do is point you to the best documentation that exists. Here are some hints:

* The authors strongly suggest getting a copy of the Tk man pages. Specifically, the man pages in the man Ndirectory are most useful. The man3 man pages describe the C interface to the Tk library and thus are not especially helpful for script writers.
* Addison-Wesley publishes a book called Tcl and the Tk Toolkit by John Ousterhout (ISBN 0-201-63337-X) which is a good introduction to Tcl and Tk for the novice. The book is not exhaustive, and for many details it defers to the man pages.
* tkinter/\_\_init\_\_.py is a last resort for most, but can be a good place to go when nothing else makes sense.

The class hierarchy looks complicated, but in actual practice, application programmers almost always refer to the classes at the very bottom of the hierarchy.

Notes:

* These classes are provided for the purposes of organizing certain functions under one namespace. They aren’t meant to be instantiated independently.
* The [Tk](https://docs.python.org/3/library/tkinter.html" \l "tkinter.Tk" \o "tkinter.Tk) class is meant to be instantiated only once in an application. Application programmers need not instantiate one explicitly, the system creates one whenever any of the other classes are instantiated.
* The Widget class is not meant to be instantiated, it is meant only for subclassing to make “real” widgets (in C++, this is called an ‘abstract class’).

To make use of this reference material, there will be times when you will need to know how to read short passages of Tk and how to identify the various parts of a Tk command. (See section [Mapping Basic Tk into Tkinter](https://docs.python.org/3/library/tkinter.html#tkinter-basic-mapping) for the [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) equivalents of what’s below.)

Tk scripts are Tcl programs. Like all Tcl programs, Tk scripts are just lists of tokens separated by spaces. A Tk widget is just its *class*, the *options* that help configure it, and the *actions* that make it do useful things.

To make a widget in Tk, the command is always of the form:

classCommand newPathname option

* classCommand: denotes which kind of widget to make (a button, a label, a menu…)
* newPathname: is the new name for this widget. All names in Tk must be unique. To help enforce this, widgets in Tk are named with *pathnames*, just like files in a file system. The top level widget, the *root*, is called . (period) and children are delimited by more periods. For example, .myApp.controlPanel.okButton might be the name of a widget.
* options: configure the widget’s appearance and in some cases, its behavior. The options come in the form of a list of flags and values. Flags are preceded by a ‘-‘, like Unix shell command flags, and values are put in quotes if they are more than one word.

Once created, the pathname to the widget becomes a new command. This new widget command is the programmer’s handle for getting the new widget to perform some action. In C, you’d express this as someAction(fred, someOptions), in C++, you would express this as fred.someAction(someOptions), and in Tk, you say:

Note that the object name, .fred, starts with a dot.

As you’d expect, the legal values for someAction will depend on the widget’s class: .fred disable works if fred is a button (fred gets greyed out), but does not work if fred is a label (disabling of labels is not supported in Tk).

The legal values of someOptions is action dependent. Some actions, like disable, require no arguments, others, like a text-entry box’s delete command, would need arguments to specify what range of text to delete.

## 4.1.9. How Tk and Tkinter are Related

## From the top down: Your App Here (Python) A Python application makes a [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) call.

## tkinter (Python Package):

This call (say, for example, creating a button widget), is implemented in the [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) package, which is written in Python. This Python function will parse the commands and the arguments and convert them into a form that makes them look as if they had come from a Tk script instead of a Python script.

\_tkinter (C): These commands and their arguments will be passed to a C function in the \_tkinter - note the underscore - extension module.

Tk Widgets (C and Tcl): This C function is able to make calls into other C modules, including the C functions that make up the Tk library. Tk is implemented in C and some Tcl. The Tcl part of the Tk widgets is used to bind certain default behaviors to widgets, and is executed once at the point where the Python [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) package is imported. (The user never sees this stage).

Tk (C): The Tk part of the Tk Widgets implement the final mapping to …

Xlib (C): the Xlib library to draw graphics on the screen.

For a complete explanation of a given option and its behavior, see the Tk man pages for the widget in question.

Note that the man pages list “STANDARD OPTIONS” and “WIDGET SPECIFIC OPTIONS” for each widget. The former is a list of options that are common to many widgets, the latter are the options that are idiosyncratic to that particular widget. The Standard Options are documented on the options(3) man page.

No distinction between standard and widget-specific options is made in this document. Some options don’t apply to some kinds of widgets. Whether a given widget responds to a particular option depends on the class of the widget; buttons have a command option, labels do not.

The options supported by a given widget are listed in that widget’s man page, or can be queried at runtime by calling the config() method without arguments, or by calling the keys() method on that widget. The return value of these calls is a dictionary whose key is the name of the option as a string (for example, ’relief’) and whose values are 5-tuples.

Some options, like bg are synonyms for common options with long names (bg is shorthand for “background”). Passing the config() method the name of a shorthand option will return a 2-tuple, not 5-tuple. The 2-tuple passed back will contain the name of the synonym and the “real” option (such as ('bg', 'background')).

4.1.9. The Packer

The packer is one of Tk’s geometry-management mechanisms. Geometry managers are used to specify the relative positioning of the positioning of widgets within their container - their mutual master. In contrast to the more cumbersome placer (which is used less commonly, and we do not cover here), the packer takes qualitative relationship specification - above, to the left of, filling, etc - and works everything out to determine the exact placement coordinates for you.

The size of any master widget is determined by the size of the “slave widgets” inside. The packer is used to control where slave widgets appear inside the master into which they are packed. You can pack widgets into frames, and frames into other frames, in order to achieve the kind of layout you desire. Additionally, the arrangement is dynamically adjusted to accommodate incremental changes to the configuration, once it is packed.

Note that widgets do not appear until they have had their geometry specified with a geometry manager. It’s a common early mistake to leave out the geometry specification, and then be surprised when the widget is created but nothing appears. A widget will appear only after it has had, for example, the packer’s pack() method applied to it.

The pack() method can be called with keyword-option/value pairs that control where the widget is to appear within its container, and how it is to behave when the main application window is resized. Here are some examples:

### 4.1.10. The Window Manager

In Tk, there is a utility command, wm, for interacting with the window manager. Options to the wm command allow you to control things like titles, placement, icon bitmaps, and the like. In [tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter), these commands have been implemented as methods on the Wm class. Toplevel widgets are subclassed from the Wm class, and so can call the Wm methods directly.

To get at the toplevel window that contains a given widget, you can often just refer to the widget’s master. Of course if the widget has been packed inside of a frame, the master won’t represent a toplevel window. To get at the toplevel window that contains an arbitrary widget, you can call the \_root() method. This method begins with an underscore to denote the fact that this function is part of the implementation, and not an interface to Tk functionality.

### 4.1.11. Images

Images of different formats can be created through the corresponding subclass of tkinter.Image:

* BitmapImage for images in XBM format.
* PhotoImage for images in PGM, PPM, GIF and PNG formats. The latter is supported starting with Tk 8.6.

Either type of image is created through either the file or the data option (other options are available as well).

The image object can then be used wherever an image option is supported by some widget (e.g. labels, buttons, menus). In these cases, Tk will not keep a reference to the image. When the last Python reference to the image object is deleted, the image data is deleted as well, and Tk will display an empty box wherever the image was used.

4.2. Speech recognition

Speech recognition is the process of converting spoken words to text. Python supports many speech recognition engines and APIs, including Google Speech Engine, Google Cloud Speech API, Microsoft Bing Voice Recognition and IBM Speech to Text.

## Installation

A library that helps is named “SpeechRecognition”. You should install it with pyenv, pipenv or virtualenv. You can also install it system wide:

The SpeechRecognition module depends on pyaudio, you can install them from your packagemanager.

The first component of speech recognition is, of course, speech. Speech must be converted from physical sound to an electrical signal with a microphone, and then to digital data with an analog-to-digital converter. Once digitized, several models can be used to transcribe the audio to text.

Most modern speech recognition systems rely on what is known as a [Hidden Markov Model](https://en.wikipedia.org/wiki/Hidden_Markov_model)(HMM). This approach works on the assumption that a speech signal, when viewed on a short enough timescale (say, ten milliseconds), can be reasonably approximated as a stationary process—that is, a process in which statistical properties do not change over time.

In a typical HMM, the speech signal is divided into 10-millisecond fragments. The power spectrum of each fragment, which is essentially a plot of the signal’s power as a function of frequency, is mapped to a vector of real numbers known as [cepstral](https://en.wikipedia.org/wiki/Cepstrum) coefficients. The dimension of this vector is usually small—sometimes as low as 10, although more accurate systems may have dimension 32 or more. The final output of the HMM is a sequence of these vectors.

To decode the speech into text, groups of vectors are matched to one or more [phonemes](https://en.wikipedia.org/wiki/Phoneme)—a fundamental unit of speech. This calculation requires training, since the sound of a phoneme varies from speaker to speaker, and even varies from one utterance to another by the same speaker. A special algorithm is then applied to determine the most likely word (or words) that produce the given sequence of phonemes.

One can imagine that this whole process may be computationally expensive. In many modern speech recognition systems, neural networks are used to simplify the speech signal using techniques for feature transformation and dimensionality reduction before HMM recognition. Voice activity detectors (VADs) are also used to reduce an audio signal to only the portions that are likely to contain speech. This prevents the recognizer from wasting time analyzing unnecessary parts of the signal.

Fortunately, as a Python programmer, you don’t have to worry about any of this. A number of speech recognition services are available for use online through an API, and many of these services offer [Python SDKs](https://realpython.com/api-integration-in-python/).

## Picking a Python Speech Recognition Package

A handful of packages for speech recognition exist on PyPI. A few of them include:

* [apiai](https://pypi.org/project/apiai/)
* [assemblyai](https://pypi.org/project/assemblyai/)
* [google-cloud-speech](https://pypi.org/project/google-cloud-speech/)
* [pocketsphinx](https://pypi.org/project/pocketsphinx/)
* [SpeechRecognition](https://pypi.org/project/SpeechRecognition/)
* [watson-developer-cloud](https://pypi.org/project/watson-developer-cloud/)
* [wit](https://pypi.org/project/wit/)

Some of these packages—such as wit and apiai—offer built-in features, like natural language processing for identifying a speaker’s intent, which go beyond basic speech recognition. Others, like google-cloud-speech, focus solely on speech-to-text conversion.

There is one package that stands out in terms of ease-of-use: SpeechRecognition.

Recognizing speech requires audio input, and SpeechRecognition makes retrieving this input really easy. Instead of having to build scripts for accessing microphones and processing audio files from scratch, SpeechRecognition will have you up and running in just a few minutes.

The SpeechRecognition library acts as a wrapper for several popular speech APIs and is thus extremely flexible. One of these—the Google Web Speech API—supports a default API key that is hard-coded into the SpeechRecognition library. That means you can get off your feet without having to sign up for a service.

The flexibility and ease-of-use of the SpeechRecognition package make it an excellent choice for any Python project. However, support for every feature of each API it wraps is not guaranteed. You will need to spend some time researching the available options to find out if SpeechRecognition will work in your particular case.

So, now that you’re convinced you should try out SpeechRecognition, the next step is getting it installed in your environment.

## Installing SpeechRecognition

SpeechRecognition is compatible with Python 2.6, 2.7 and 3.3+, but requires some [additional installation steps for Python 2](https://github.com/Uberi/speech_recognition#requirements). For this tutorial, I’ll assume you are using Python 3.3+.

You can install SpeechRecognition from a terminal with pip:

$ pip install SpeechRecognition

Once installed, you should verify the installation by opening an interpreter session and typing:

>>>

>>> import speech\_recognition as sr

>>> sr.\_\_version\_\_

'3.8.1'

**Note:** The version number you get might vary. Version 3.8.1 was the latest at the time of writing.

Go ahead and keep this session open. You’ll start to work with it in just a bit.

SpeechRecognition will work out of the box if all you need to do is work with existing audio files. Specific use cases, however, require a few dependencies. Notably, the PyAudio package is needed for capturing microphone input.

You’ll see which dependencies you need as you read further. For now, let’s dive in and explore the basics of the package.

## The Recognizer Class

All of the magic in SpeechRecognition happens with the Recognizer class.

The primary purpose of a Recognizer instance is, of course, to recognize speech. Each instance comes with a variety of settings and functionality for recognizing speech from an audio source.

Creating a Recognizer instance is easy. In your current interpreter session, just type:

>>>

>>> r = sr.Recognizer()

Each Recognizer instance has seven methods for recognizing speech from an audio source using various APIs. These are:

* recognize\_bing(): [Microsoft Bing Speech](https://azure.microsoft.com/en-us/services/cognitive-services/speech/)
* recognize\_google(): [Google Web Speech API](https://w3c.github.io/speech-api/speechapi.html)
* recognize\_google\_cloud(): [Google Cloud Speech](https://cloud.google.com/speech/) - requires installation of the google-cloud-speech package
* recognize\_houndify(): [Houndify](https://www.houndify.com/) by SoundHound
* recognize\_ibm(): [IBM Speech to Text](https://www.ibm.com/watson/services/speech-to-text/)
* recognize\_sphinx(): [CMU Sphinx](https://cmusphinx.github.io/) - requires installing PocketSphinx
* recognize\_wit(): [Wit.ai](https://wit.ai/)

Of the seven, only recognize\_sphinx() works offline with the CMU Sphinx engine. The other six all require an internet connection.

A full discussion of the features and benefits of each API is beyond the scope of this tutorial. Since SpeechRecognition ships with a default API key for the Google Web Speech API, you can get started with it right away. For this reason, we’ll use the Web Speech API in this guide. The other six APIs all require authentication with either an API key or a username/password combination. For more information, consult the SpeechRecognition [docs](https://github.com/Uberi/speech_recognition/blob/master/reference/library-reference.rst).

## Working With Audio Files

Before you continue, you’ll need to download an audio file. The one I used to get started, “harvard.wav,” can be found [here](https://github.com/realpython/python-speech-recognition). Make sure you save it to the same directory in which your Python interpreter session is running.

SpeechRecognition makes working with audio files easy thanks to its handy AudioFile class. This class can be initialized with the path to an audio file and provides a context manager interface for reading and working with the file’s contents.

### Supported File Types

Currently, SpeechRecognition supports the following file formats:

* WAV: must be in PCM/LPCM format
* AIFF
* AIFF-C
* FLAC: must be native FLAC format; OGG-FLAC is not supported

If you are working on x-86 based Linux, macOS or Windows, you should be able to work with FLAC files without a problem. On other platforms, you will need to install a FLAC encoder and ensure you have access to the flac command line tool. You can find more information [here](https://xiph.org/flac/) if this applies to you.

### Using record() to Capture Data From a File

Type the following into your interpreter session to process the contents of the “harvard.wav” file:

>>>

>>> harvard = sr.AudioFile('harvard.wav')

>>> with harvard as source:

... audio = r.record(source)

...

The context manager opens the file and reads its contents, storing the data in an AudioFileinstance called source. Then the record() method records the data from the entire file into an AudioData instance. You can confirm this by checking the type of audio:

>>>

>>> type(audio)

<class 'speech\_recognition.AudioData'>

You can now invoke recognize\_google() to attempt to recognize any speech in the audio. Depending on your internet connection speed, you may have to wait several seconds before seeing the result.

>>>

>>> r.recognize\_google(audio)

'the stale smell of old beer lingers it takes heat

to bring out the odor a cold dip restores health and

zest a salt pickle taste fine with ham tacos al

Pastore are my favorite a zestful food is the hot

cross bun'

Congratulations! You’ve just transcribed your first audio file!

If you’re wondering where the phrases in the “harvard.wav” file come from, they are examples of Harvard Sentences. These phrases were published by the IEEE in 1965 for use in speech intelligibility testing of telephone lines. They are still used in VoIP and cellular testing today.

The Harvard Sentences are comprised of 72 lists of ten phrases. You can find freely available recordings of these phrases on the [Open Speech Repository](http://www.voiptroubleshooter.com/open_speech/index.html) website. Recordings are available in English, Mandarin Chinese, French, and Hindi. They provide an excellent source of free material for testing your code.

### Capturing Segments With offset and duration

What if you only want to capture a portion of the speech in a file? The record() method accepts a duration keyword argument that stops the recording after a specified number of seconds.

For example, the following captures any speech in the first four seconds of the file:

>>>

>>> with harvard as source:

... audio = r.record(source, duration=4)

...

>>> r.recognize\_google(audio)

'the stale smell of old beer lingers'

The record() method, when used inside a with block, always moves ahead in the file stream. This means that if you record once for four seconds and then record again for four seconds, the second time returns the four seconds of audio after the first four seconds.

>>>

>>> with harvard as source:

... audio1 = r.record(source, duration=4)

... audio2 = r.record(source, duration=4)

...

>>> r.recognize\_google(audio1)

'the stale smell of old beer lingers'

>>> r.recognize\_google(audio2)

'it takes heat to bring out the odor a cold dip'

Notice that audio2 contains a portion of the third phrase in the file. When specifying a duration, the recording might stop mid-phrase—or even mid-word—which can hurt the accuracy of the transcription. More on this in a bit.

In addition to specifying a recording duration, the record() method can be given a specific starting point using the offset keyword argument. This value represents the number of seconds from the beginning of the file to ignore before starting to record.

To capture only the second phrase in the file, you could start with an offset of four seconds and record for, say, three seconds.

>>>

>>> with harvard as source:

... audio = r.record(source, offset=4, duration=3)

...

>>> recognizer.recognize\_google(audio)

'it takes heat to bring out the odor'

The offset and duration keyword arguments are useful for segmenting an audio file if you have prior knowledge of the structure of the speech in the file. However, using them hastily can result in poor transcriptions. To see this effect, try the following in your interpreter:

>>>

>>> with harvard as source:

... audio = r.record(source, offset=4.7, duration=2.8)

...

>>> recognizer.recognize\_google(audio)

'Mesquite to bring out the odor Aiko'

By starting the recording at 4.7 seconds, you miss the “it t” portion a the beginning of the phrase “it takes heat to bring out the odor,” so the API only got “akes heat,” which it matched to “Mesquite.”

Similarly, at the end of the recording, you captured “a co,” which is the beginning of the third phrase “a cold dip restores health and zest.” This was matched to “Aiko” by the API.

There is another reason you may get inaccurate transcriptions. Noise! The above examples worked well because the audio file is reasonably clean. In the real world, unless you have the opportunity to process audio files beforehand, you can not expect the audio to be noise-free.

### VADER Sentiment Analysis

[**VADER**](https://github.com/cjhutto/vaderSentiment) (**Valence Aware Dictionary and sEntiment Reasoner**) is a lexicon and rule-based sentiment analysis tool that is**specifically attuned to sentiments expressed in social media**. VADER uses a combination of A [sentiment lexicon](http://livernspleen.com/wp-content/uploads/2013/09/social-media-tool-as-a-learning-resource.pdf) is a list of lexical features (e.g., words) which are generally labelled according to their semantic orientation as either positive or negative.

VADER has been found to be quite successful when dealing with social media texts, NY Times editorials, movie reviews, and product reviews. This is because VADER not only tells **about** the Positivity and Negativity score but also tells us about **how positive or negative a sentiment is**.

It is fully open-sourced under the [MIT License](http://choosealicense.com/). The developers of VADER have used [Amazon’s Mechanical Turk](https://www.mturk.com/) to get most of their ratings, You can find complete details on their [Github Page](https://github.com/cjhutto/vaderSentiment" \t "_blank).

[methods and process approach overview](http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf)

### Advantages of using VADER

VADER has a lot of advantages over traditional methods of Sentiment Analysis, including:

* It works exceedingly well on social media type text, yet readily generalizes to multiple domains
* It**doesn’t require any training data** but is constructed from a generalizable, valence-based, human-curated gold standard sentiment lexicon
* It is fast enough to be used online with streaming data, and
* It does not severely suffer from a speed-performance tradeoff.

The source of this article is a very easy to read paper published by the creaters of VADER library.You can read the paper [here](http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf).

Enough of talking. Let us now see practically how does VADER analysis work for which we will have install the library first.

### Installation

The simplest way is to use the command line to do an installation from [[PyPI]](https://pypi.python.org/pypi/vaderSentiment)using pip. Check their [Github repository](https://github.com/cjhutto/vaderSentiment" \t "_blank) for the detailed explanation.

**> pip install vaderSentiment**

Once VADER is installed let us call the **SentimentIntensityAnalyser** object,

**from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer**

**analyser = SentimentIntensityAnalyzer()**

### Working & Scoring

Let us test our first sentiment using VADER now. We will use the **polarity\_scores()** method to obtain the polarity indices for the given sentence.

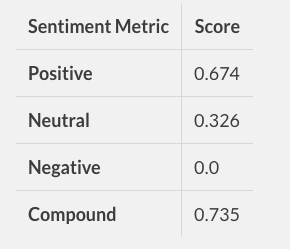
def sentiment\_analyzer\_scores(sentence):  
 score = analyser.polarity\_scores(sentence)  
 print("{:-<40} {}".format(sentence, str(score)))

Let us check how VADER performs on a given review:

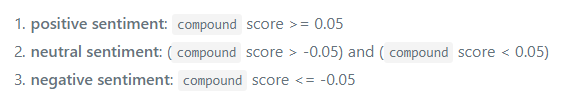
**sentiment\_analyzer\_scores("The phone is super cool.")**

**The phone is super cool----------------- {'neg': 0.0, 'neu': 0.326, 'pos': 0.674, 'compound': 0.7351}**

Putting in a Tabular form:



* The Positive, Negative and Neutral scores represent the proportion of text that falls in these categories. This means our sentence was rated as 67% Positive, 33% Neutral and 0% Negative. Hence all these should add up to 1.
* The Compound score is a metric that calculates the sum of all the [lexicon ratings](https://github.com/cjhutto/vaderSentiment/blob/master/vaderSentiment/vader_lexicon.txt) which have been normalized between -1(most extreme negative) and +1 (most extreme positive). In the case above, lexicon ratings for andsupercool are 2.9and respectively1.3. The compound score turns out to be 0.75 , denoting a very high positive sentiment.



compound score metric

read [here](https://github.com/cjhutto/vaderSentiment#about-the-scoring) for more details on VADER scoring methodology.

VADER analyses sentiments primarily based on certain key points:

* **Punctuation: T**he use of an exclamation mark**(!)**, increases the magnitude of the intensity without modifying the semantic orientation. For example, “The food here is good!” is more intense than “The food here is good.” and an increase in the number of **(!)**, increases the magnitude accordingly

MoviePy:

MoviePy is a Python module for video editing, which can be used for basic operations (like cuts, concatenations, title insertions), video compositing (a.k.a. non-linear editing), video processing, or to create advanced effects. It can read and write the most common video formats, including GIF.

## Installation

**Method with pip:** if you have pip installed, just type this in a terminal (it will install ez\_setup if you don’t already have it)

(sudo) pip install moviepy

If you have neither setuptools nor ez\_setup installed the command above will fail, is this case type this before installing:

(sudo) pip install ez\_setup

**Method by hand:** download the sources, either on [PyPI](https://pypi.python.org/pypi/moviepy) or (if you want the development version) on [Github](https://github.com/Zulko/moviepy), unzip everything in one folder, open a terminal and type

(sudo) python setup**.**py install

MoviePy depends on the Python modules [Numpy](https://www.scipy.org/install.html), [imageio](https://imageio.github.io/), [Decorator](https://pypi.python.org/pypi/decorator), and [tqdm](https://pypi.python.org/pypi/tqdm), which will be automatically installed during MoviePy’s installation. It should work on Windows/Mac/Linux, with Python 2.7+ and 3 ; if you have trouble installing MoviePy or one of its dependencies, please provide feedback !

MoviePy depends on the software FFMPEG for video reading and writing. You don’t need to worry about that, as FFMPEG should be automatically downloaded/installed by ImageIO during your first use of MoviePy (it takes a few seconds). If you want to use a specific version of FFMPEG, you can set the FFMPEG\_BINARY environment variable See moviepy/config\_defaults.py

# Download and Installation

## Installation

**Method with pip:** if you have pip installed, just type this in a terminal (it will install ez\_setup if you don’t already have it)

(sudo) pip install moviepy

If you have neither setuptools nor ez\_setup installed the command above will fail, is this case type this before installing:

(sudo) pip install ez\_setup

**Method by hand:** download the sources, either on [PyPI](https://pypi.python.org/pypi/moviepy) or (if you want the development version) on [Github](https://github.com/Zulko/moviepy), unzip everything in one folder, open a terminal and type

(sudo) python setup**.**py install

MoviePy depends on the Python modules [Numpy](https://www.scipy.org/install.html), [imageio](https://imageio.github.io/), [Decorator](https://pypi.python.org/pypi/decorator), and [tqdm](https://pypi.python.org/pypi/tqdm), which will be automatically installed during MoviePy’s installation. It should work on Windows/Mac/Linux, with Python 2.7+ and 3 ; if you have trouble installing MoviePy or one of its dependencies, please provide feedback !

MoviePy depends on the software FFMPEG for video reading and writing. You don’t need to worry about that, as FFMPEG should be automatically downloaded/installed by ImageIO during your first use of MoviePy (it takes a few seconds). If you want to use a specific version of FFMPEG, you can set the FFMPEG\_BINARY environment variable See moviepy/config\_defaults.py for details.

### Other optional but useful dependencies

[ImageMagick](https://www.imagemagick.org/script/index.php) is not strictly required, only if you want to write texts. It can also be used as a backend for GIFs but you can do GIFs with MoviePy without ImageMagick.

Once you have installed it, ImageMagick will be automatically detected by MoviePy, **except on Windows !**. Windows user, before installing MoviePy by hand, go into the moviepy/config\_defaults.py file and provide the path to the ImageMagick binary called convert. It should look like this

IMAGEMAGICK\_BINARY **=** "C:\\Program Files\\ImageMagick\_VERSION\\convert.exe"

You can also set the IMAGEMAGICK\_BINARY environment variable See moviepy/config\_defaults.pyfor details.

[PyGame](https://www.pygame.org/download.shtml) is needed for video and sound previews (useless if you intend to work with MoviePy on a server but really essential for advanced video editing by hand).

For advanced image processing you will need one or several of these packages. For instance using the method clip.resize requires that at least one of Scipy, PIL, Pillow or OpenCV are installed.

* The Python Imaging Library (PIL) or, better, its branch [Pillow](https://pillow.readthedocs.org/en/latest/) .
* [Scipy](https://www.scipy.org/) (for tracking, segmenting, etc.), and can be used for resizing video clips if PIL and OpenCV aren’t installed on your computer.
* [Scikit Image](http://scikit-image.org/download.html) may be needed for some advanced image manipulation.
* [OpenCV 2.4.6](https://sourceforge.net/projects/opencvlibrary/files/) or more recent (provides the package cv2) or more recent may be needed for some advanced image manipulation.

If you are on linux, these softwares will surely be in your report.

**Chapter – 5**

**Sample Code**

1. Script for the interface of the application

Script:

from tkinter import \*

from ver import first

from sentimentanalyzer import second

global k

def printtext():

global e

string=e.get()

l=first(string)

k=second()

root=Tk()

root.geometry("500x100+300+300")

label = Message(root, text=l, width=500, relief=RAISED)

label.pack()

label=Message(root,text=k,width=100,relief=RAISED)

label.pack()

root.mainloop()

top=Tk()

top.geometry("500x100+300+300")

top.title('Video sentimental analyser')

label = Message(top, text="Enter path", width=100, relief=RAISED)

label.pack()

e=Entry(top,width=50)

e.pack()

e.focus\_set()

b=Button(top,text='processing',command=printtext)

b.pack(side='bottom')

top.mainloop()

1. Script to convert Video to Text

Script:

import moviepy.editor as mp

import speech\_recognition as sr

from os import path

def first(file\_path):

clip = mp.VideoFileClip(file\_path).subclip(100,250)

clip.audio.write\_audiofile("theaudio.wav")

AUDIO\_FILE = path.join(path.dirname(path.realpath(\_\_file\_\_)), "theaudio.wav")

r=sr.Recognizer()

with sr.AudioFile(AUDIO\_FILE) as source:

audio = r.record(source)

try:

file=open(r"C:\Users\hritik\Desktop\input.txt","w+")

k=str(r.recognize\_google(audio,language="en-US"))

file.write(k)

print("Google Speech Recognition thinks you said " +k)

file.close()

return k

except sr.UnknownValueError:

print("Google Speech Recognition could not understand audio")

except sr.RequestError as e:

print("Could not request results from Google Speech Recognition service; {0}".format(e))

3. Script for Sentimental Analysis of the converted Text File

Script:

from textblob import TextBlob

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

def second():

analyzer=SentimentIntensityAnalyzer()

pos\_count=0

pos\_correct=0

file= open(r"C:\Users\hritik\Desktop\input.txt","r")

with file as f:

for line in f.read().split('\n'):

analysis=analyzer.polarity\_scores(line)

if analysis['compound'] > 0.5:

pos\_correct+=1

pos\_count+=1

file.close()

file1= open(r"C:\Users\hritik\Desktop\input.txt","r")

neg\_count=0

neg\_correct=0

with file1 as f:

for line in f.read().split('\n'):

analysis=analyzer.polarity\_scores(line)

if analysis['compound'] <= 0.5:

neg\_correct+=1

neg\_count+=1

file1.close()

p=(pos\_correct/pos\_count\*100.0,pos\_count)

n=(neg\_correct/neg\_count\*100.0,neg\_count)

if(p>n):

return str("Video is positive")

else:

return str("Video can hurt anyones sentiments")

**Chapter - 5**

**Reference/Bibliography**

* <https://www.quora.com/How-do-I-perform-sentiment-analysis-on-YouTube-videos>
* <https://www.github.com>
* <https://www.wikipedia.com>
* <https://ieeexplore.ieee.org/document/8120658/>
* https://www.researchgate.net/publication/321408074\_YouTube\_video\_by\_sentiment\_analysis
* <https://stackoverflow.com/questions/44773553/what-i-can-use-to-process-video-sentiment-analysis-on-a-video-stream>
* Natural Language Processing with python by Edward Loper, Steven Bird, Ewan Klein.