.

AUTOMATIC HIGHLIGHT GENERATION USING PYTHON

by

Kamesh Shekhar Prasad

(Registration Number: 17352214)

Project report submitted in partial fulfilment of the requirements for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF ENGINEERING & TECHNOLOGY

PONDICHERRY UNIVERSITY, KARAIKAL CAMPUS

June 2020

BONAFIDE CERTIFICATE

This is to certify that this major project work entitled **“Automatic Highlight Generation Using Python”** is a bonafide record of work done by **Mr. Kamesh Shekhar Prasad** in partial fulfilment for the degree of Master of Computer Applications of Pondicherry University.

This work has not been submitted elsewhere for the award of any other degree to the best of our knowledge.

INTERNAL GUIDE

Mr. S. Balaji

Guest Faculty

Department of Computer Science

School of Engineering

Pondicherry University, Karaikal Campus

Karaikal – 609 602

HEAD OF THE DEPARTMENT

Dr. G. Kumaravelan

Assistant Professor and Head (i/c)

Department of Computer Science

School of Engineering

Pondicherry University, Karaikal Campus

Karaikal – 609 602

Submitted for the Viva-Voce Examination held on ________________

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

This is one of the greatest and the most pleasant experiences of my life till now. Though I faced a series of mishap during the period of my project, with the help of my guide **Mr. S. Balaji**, Guest Faculty of Pondicherry University, Karaikal Campus, I have done it on my own. From the dawn to end I must make thanks to God since there couldn’t be a bit move without God’s grace.

I am highly in dept to **Mr. S. Balaji**, Guest Faculty of Pondicherry University, Karaikal Campus, for his guidance and constant supervision as well as for providing necessary information regarding the project and also for his support in completing the project.

I am highly in dept to **Dr. G. Kumaravelan**, Head of the Department of Computer Science, for his valuable support and encouragement leading to successful completion of my report. I would like to express my gratitude towards all the faculty members and Ph. D. Scholars of Department of Computer Science, Pondicherry University, Karaikal Campus for their kind corporation and encouragement.

Also, my thanks and appreciation to my colleges in developing the project and the people who have willingly helped me out with their abilities. My parents always stand with me and sail through me in all my hard times and I am always grateful to them.

KAMESH SHEKHAR PRASAD

SYNOPSIS

Cricket is most famous sport in India and played in almost all parts of the country. So, being a die-hard cricket fan, I decided to automate the process of highlights extractions from a full match cricket video. Nevertheless, the same idea can be applied to other sports as well.

Developing a program for automatic highlight generation is an approach to be used to summarization of full-length video. This proposal gives an overview of the work that has been done to make highlight of input sports video**.**

The present work is a beginning to a higher goal of reducing task of persons involved in video editing industry.

TABLE OF CONTENTS

1. INTRODUCTION

We have all seen highlights of sports matches at some point. Even if you don’t have an inclination towards sports, you will have come across highlights on television while sitting in a restaurant, lounging in a hotel, etc.

Today is the world of sports. Every where many sports are played. In India cricket is played most. Indians worship cricketers as god. So, everyone wants to see each cricket match but due to lack of time everyone is not able to see whole match. So many people wish to see highlight after their daily life ends.

But finding highlight is a hectic task. We have to wait for hours for seeing highlights in a specific news channel. So, I a person know python he can generate his own highlight without help of any other using some python packages.

CRICKET

Cricket is considered as one of the most important and loved sports of India. Hockey is national game only on paper but in hearts of every Indian only cricket rules the world In cricket full match contains special moments like fighting between players, lots of appeals of bowlers, waiting for run out, fours, sixes, wickets, singing national anthem, post-match presentation ceremony, distributing prizes to players and other staffs, showing special moments of spectators and a lot of various things and exciting moments.

But in highlight we only want to reduce time of cricket match as we know that full cricket match takes too much time. Test match takes 8 hours per day for five consecutive days. One day international takes almost 8 hours. But now a days twenty over match takes less time. But a busy person wants to see in less possible time.

There are different ways through which we can create highlight of a cricket match or any sports video. Like we can use any video editing software but it requires lots of money and time. We have to purchase that software for using it.

So, I want to develop a simple application through which people can enjoy highlights without wasting money.

The video editing industry requires lots of hard work for creating highlights. The unedited version even captures uninteresting features like defences, leaves, wide balls byes, etc. But we have to perfect in every work. I want to develop highlight which doesn’t contain any unnecessary things like appeals of bowlers, fighting’s between players and a lot of other things. So, using python we can develop main things of cricket not only unnecessary things. Using python packages, it can be easily developed.

Extracting highlights manually from a full match video requires a lot of human effort. It is a time-consuming process. A person who is working in video editing company has to see whole video then he has to analyse which part should be kept and which part should be removed. It requires lots of unnecessary time. It is also memory-hogging to store a full match video. So, extracting highlight automatically is high demand of these video industry as it will save their lots of time and their money given as salary to their employees. So, I want to reduce the task of video industry by creating automatic generation of highlight where user have to only input video and automatically highlights will be generated.

2. PROBLEM DEFINTION AND FEASIBILITY ANALYSIS

Cricket is the most famous sport in India and played in almost all parts of the country**.** So, being a die-hard cricket fan, I decided to automate the process of highlights extraction from a full match cricket video. Nevertheless, the same idea can be applied to other sports as well.

For this project I am taking a you tube URL and downloading any cricket video or any sports video. Suppose I downloaded first six overs (Powerplay) of the semi-final match between India and Australia at the T20 World Cup in 2007. SO, after downloading I extracted audio from video file and then using librosa package in python I loaded my whole audio file in my python script. Then I break whole video into 5 seconds of chunks. Then I am calculating energy of every chunk. Then I am calculating short time energy of every chunk. Then using matplotlib package in python a graph is shown. Then I choose the extreme value as the threshold since we are interested in the clips only when the commentator's speech and spectators' cheers are high.

3. SOFTWARE REQUIREMENT SPECIFICATION

HARDWARE REQUIREMENTS

Processor	: Intel Core i3 CPU
RAM	: 2GB (minimum)

SOFTWARE REQUIREMENTS

Operating System: Windows 8.1

Development Kit : Python 3, Jupyter Notebook

4. SYSTEM ANALYSIS

METHODOLOGIES TO BE USED IN THIS SYSTEM ANALYSIS

(1) Data Input

We are going to use www.youtube.com as our source for downloading data. Cricket video is downloaded directly using you tube using youtube-dl package. youtube-dl is a command line program to download videos from youtube.com and a few more sits. It requires the Python Interpreter version 3.6 or lower.

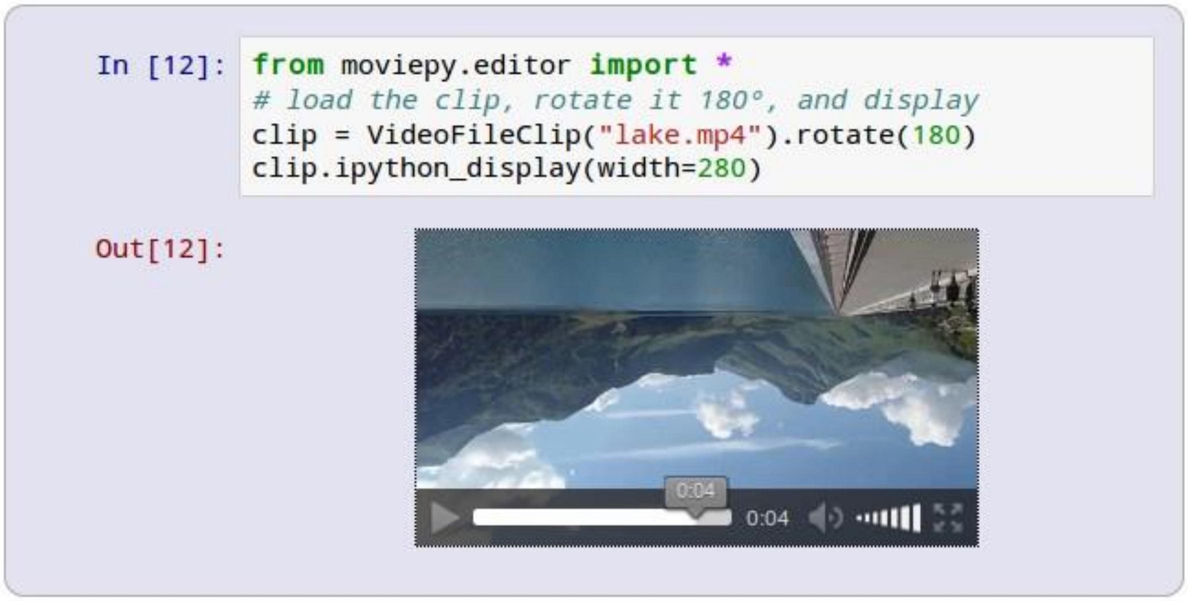
Input Video: https://www.youtube.com/ watch?v=AkrP0QtolwA/

This video is downloaded from you tube for development purpose. You can give any other sports video for generating highlights.

(2) Extracting audio

Extracting audio from input video is very easy using moviepy python library. moviepy is a Python library for video editing: cutting, concatenations, title insertions, video compositing, video processing and creation of custom effects. moviepy can read and write all the most common audio and video formats, including GIF, and runs of Windows/Mac/Linux, with Python 2.7+ and 3(or only Python 3.4+ from v.1.0). Here it is in action in an ipython notebook:

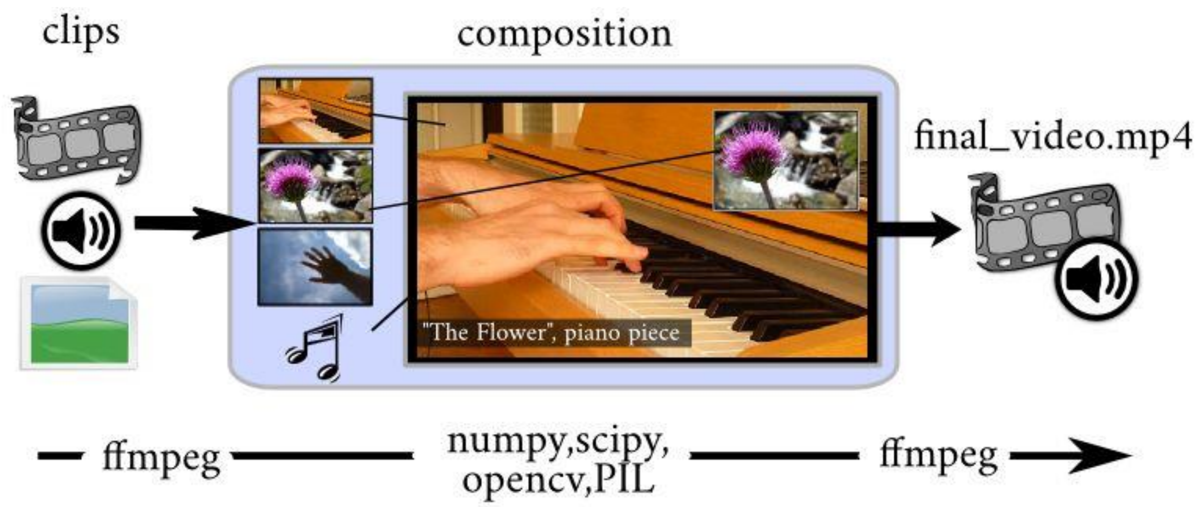
arrive at modeling the data in order to apply Machine learning. Plotting in EDA consists of Histograms, Box plot, Scatter plot and many more. It often takes much time to explore the data. Through the process of EDA, we can ask to define the problem statement or definition on our data set which is very important. Here, it involves viewing the first five rows of the data, viewing the last five rows of the data, viewing the shape of the data, check the unique labels of the target variable (Outcome) and the descriptive statistics of the data 8.



```
In [12]: from moviepy.editor import *
         # load the clip, rotate it 180°, and display
         clip = VideoFileClip("lake.mp4").rotate(180)
         clip.ipython_display(width=280)

Out[12]:
```

How moviepy Works:

moviepy uses the software ffmpeg to read and to export video and audio files. It also (optionally) uses imageMagick to generate texts and write GIF files. The Processing of the different media is ensured by Python’s fast numerical library numpy. Advanced effects and enhancements use some of Python’s numerous image processing libraries(PIL, Scikit-image, SciPy, etc.).



(3) Break the audio into chunks

For breaking the audio into chunks librosa package is used. librosa is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.

librosa package is structured as collection of submodules.

- **librosa.beat**

  Functions for estimating tempo and detecting beat events**.**

- **librosa.core**

  Core functionality includes functions to load audio from disk, compute various spectrogram representations, and a variety of commonly used tools for music analysis. For convenience, all functionality in this submodule is directly accessible from the top-level librosa.* namespace.

- **librosa.decompose**

  Functions for harmonic-percussive source separation (HPSS) and generic spectrogram decomposition using matrix decomposition methods implemented in *scikit-learn*.

- **librosa.display**

  Visualization and display routines using **matplotlib**.

- **librosa.effects**

  Time-domain audio processing, such as pitch shifting and time stretching. This submodule also provides time-domain wrappers for the *decompose* submodule.

- **librosa.feature**

  Feature extraction and manipulation. This includes low-level feature extraction, such as chromagrams, pseudo-constant-Q (log-frequency) transforms, Mel spectrogram, MFCC, and tuning estimation. Also provided are feature manipulation methods, such as delta features, memory embedding, and event-synchronous feature alignment.

- **librosa.filters**

  Filter-bank generation (chroma, pseudo-CQT, CQT, etc.). These are primarily internal functions used by other parts of *librosa*.

- **librosa.onset**

  Onset detection and onset strength computation.

- **librosa.output**

  Text- and wav-file output. *(Deprecated)*

- **librosa.segment**

  Functions useful for structural segmentation, such as recurrence matrix construction, time-lag representation, and sequentially constrained clustering.

- **librosa.sequence**

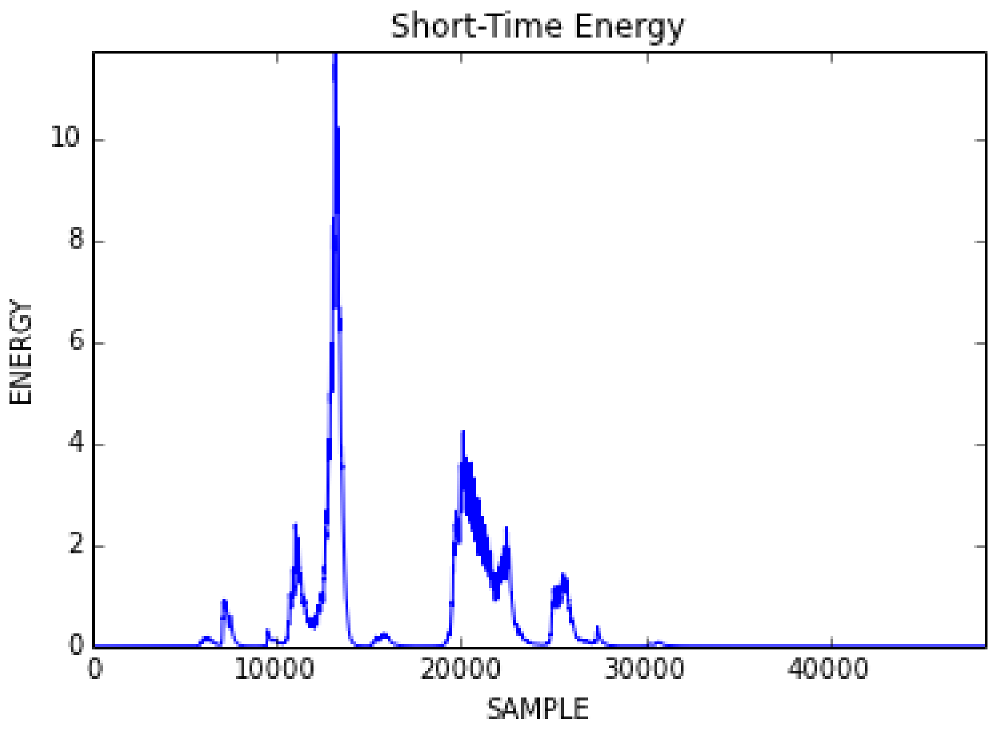
  Functions for sequential modeling. Various forms of Viterbi decoding, and helper functions for constructing transition matrices.

- **librosa.util**

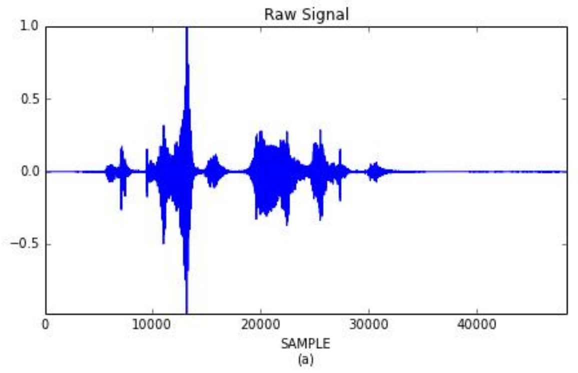
  Helper utilities (normalization, padding, centering, etc.)

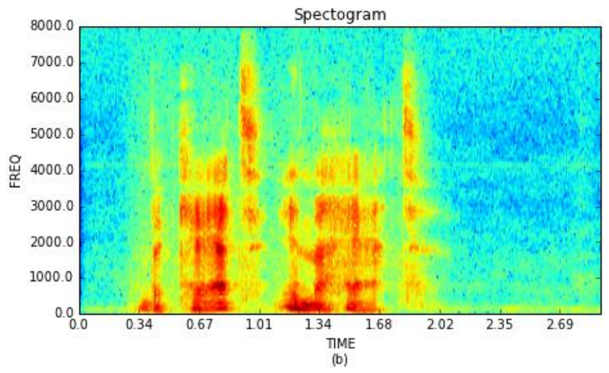
(4) Short time energy

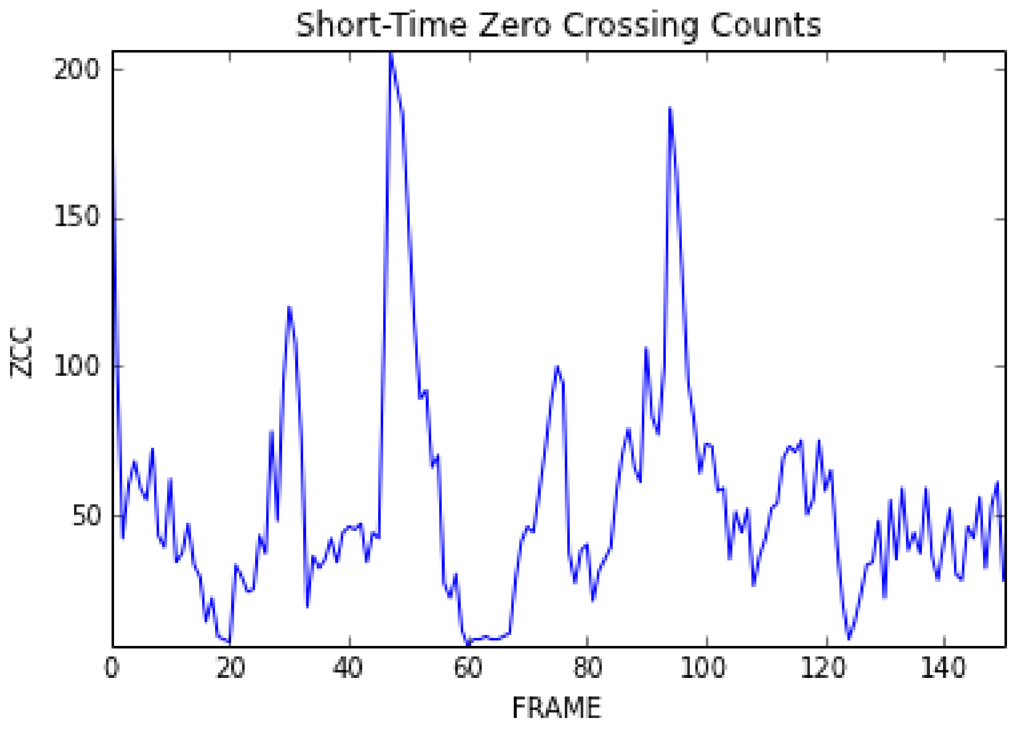
The short-time energy of speech may be computer by dividing the speech signal into frames of N samples and inserting the sum squared values of the samples in each frame. Splitting the signals into frames can be achieved by multiplying the signal by a suitable window w(n) where N= 0,1,2,…N-1 (w(n)) -> 0 for n outside the range[0, N-1].



Visualization of the Short time energy:







(5) Merge all clips using moviepy.editor:

Some of modules used:

- **Get_frame(self,t):**

  Gets a numpy array representing the RGB picture of the clip at time t or (mono or stereo) value for a sound clip.

- **is_playing(self,t):**

  If t is a time, returns true if t is between the start and the end of the clip. T can be expressed in seconds (15.35), in (min, sec), in (hour, min, sec), or as string: '01:03:05.35'. If t is a NumPy array, returns False if none of the t is in the clip, else returns a vector[b_1,b_2,b_3…] where b_i is true iff tti is in the clip.

- **Save_frame(self,filename,t=0,withmask=True)**

  Save a clip's frame to an image file.

  Saves the frame of clip corresponding to time t in 'filename'. T can be expressed in seconds (15.35), in (min,sec), in (hour, min, sec), or as a string: '01:03:05.35'.

  If withmask is True the mask is saved in the alpha layer of the picture (only works with PNGs).

- **Set_audio(self, audioclip)**

  Attach an AudioClip to the VideoClip.

  Returns a copy of the VideoClip instance, with the audio attribute set to audio, which must be an AudioClip instance.

- **set_duration(self, t, change_end=True)**

  Returns a copy of the clip, with the duration attribute set to t, which can be expressed in seconds (15.35), in (min, sec), in (hour, min, sec), or as a string: '01:03:05.35'. Also sets the duration of the mask and audio, if any, of the returned clip. If change_end is False, the start attribute of the clip will be modified in function of the duration and the present end of the clip.

- **set_end(self,t)**

  Returns a copy of the clip, with the end attribute set to t, which can be expressed in seconds (15.35), in (min, sec), in (hour, min, sec), or as a string: ’01:03:05.35’. Also sets the duration of the mask and audio, if any, of the returned clip.

- **Set_fps(self,fps)**

  Returns a copy of the clip with a new default fps for functions like write_videofile, iterframe, etc.

- **Set_make_frame(self,mf)**

  Change the clip’s get_frame.

  Returns a copy of the video clip instance, with the make_frame attribute set to mf.

- **Set_memoize(self,memorize)**

  Sets whether the clip should keep the last frame read in memory.

- **Set_pos(*a, **kw)**

  The function set_pos is deprecated and is kept temporarily for backwards compatibility. Please use the new name, set_postition, instead.

- **Set_position(self, pos, relative=False)**

  Set the clip’s postion in compositions.

  Set the position that the clip will have when included in compositions. The argument pos can be either a couple (x, y) or a function t-> (x, y). x and y mark the location of the top left corner of the clip and can be of several types.

- **Set_start(self, t, change_end=True)**

  Returns a copy of the clip, with the start attribute set to t, which can be expressed in seconds (15.35), in (min, sec), in (hour, min, sec), or as a string:’01:03:05:35’.

- **Subclip(self, t_start=0, t_end=None)**

  Returns a clip playing the content of the current clip between times t_start and t_end, which can be expressed in seconds (15.35), in (min, sec), in hour, min, sec), or as a string: ’01:03:05.35’. if t_end is not provided, it is assumed to be the duration of the clip (potentially infinite). If t_end is a negative value, it is reset to ‘’clip. Duration +t_end.’’

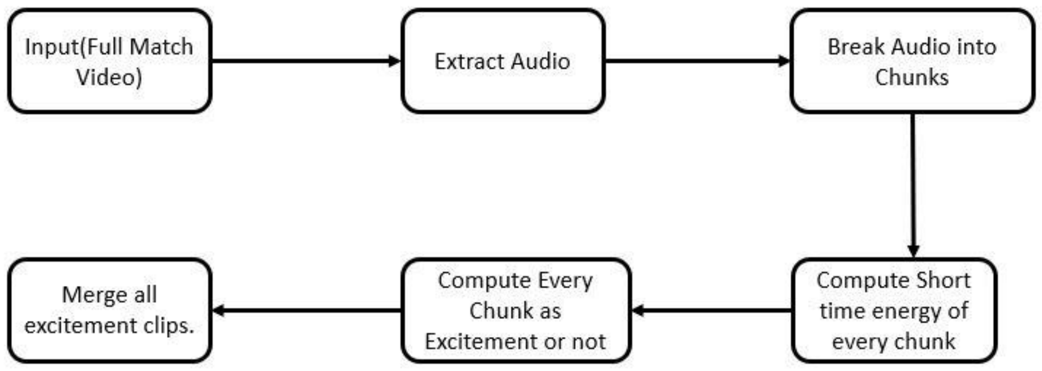
  If t_end is providing or if the clip has a duration attribute, the duration of the returned clip is set automatically.

  The mask and audio of the resulting sub clip will be sub clips of mask and audio the original clip, if they exist.
- **To_ImageClip(self, t=0, with_mask=True, duration=None)**

  Returns an Image Clip made out of the clip’s frame at time t, which can be expressed in seconds(15.35), in (min, sec), in (hour, min, sec), or as a string: ’01:03:05.35’.

5. SYSTEM DESIGN

Systems Design is an active field in which analysts repetitively learn new approaches and different techniques for building the system more effectively and efficiently. The primary objective of systems analysis and design is to improve organizational systems. This provides a basic understanding of system characteristics, system design, and its development processes. It is a good introductory guide that provides an overview of all the concepts necessary to build a system.


Input(Full Match Video)
Extract Audio
Break Audio into Chunks
Compute Short time energy of every chunk
Compute Every Chunk as Excitement or not
Merge all excitement clips.

Architecture

THE FOLLOWING ARE THE STEPS TO BE PERFORMED IN THE SYSTEM DESIGN:

1. Import the Libraries

2. Input video

3. Extract Audio

4. Break audio into chunks

5. Compute short time energy of every chunk

6. Compute every chunk as excitement or not

7. Merge all excitement clips.

THE PYTHON LIBRARIES TO BE USED HERE ARE:

youtube_dl:

youtube-dl is a command-line program to download videos from YouTube.com and a few more sites. It requires the Python interpreter, version 2.6, 2.7, or 3.2+, and it is not platform specific. It should work on your Unix box, on Windows or on macOS. It is released to the public domain, which means you can modify it, redistribute it or use it however you like.

You can configure youtube-dl by placing any supported command line option to a configuration file. On Linux and macOS, the system wide configuration file is located at /etc/youtube-dl.conf and the user wide configuration file at ~/.config/youtube-dl/config. On Windows, the user wide configuration file locations are %APPDATA%\youtube-dl\config.txt or C:\Users\<user name>\youtube-dl.conf. Note that by default configuration file may not exist so you may need to create it yourself.

You may also want to configure automatic credentials storage for extractors that support authentication (by providing login and password with --username and --password) in order not to pass credentials as command line arguments on every youtube-dl execution and prevent tracking plain text passwords in the shell command history. You can achieve this using a .netrc file on a per extractor basis.

moviepy:

moviepy (full documentation) is a Python library for video editing: cutting, concatenations, title insertions, video compositing (a.k.a. non-linear editing), video processing, and creation of custom effects. See the gallery for some examples of use.

moviepy can read and write all the most common audio and video formats, including GIF, and runs on Windows/Mac/Linux, with Python 2.7+ and 3 (or only Python 3.4+ from v.1.0). Here it is in action in an ipython notebook

moviepy depends on the Python modules numpy, imageio, Decorator, and tqdm, which will be automatically installed during moviepy's installation. The software FFMPEG should be automatically downloaded/installed (by imageio) during your first use of moviepy (installation will take a few seconds).

imageMagick is not strictly required, but needed if you want to incorporate texts. It can also be used as a backend for GIFs, though you can also create GIFs with moviepy without imageMagick.

Once you have installed imageMagick, it will be automatically detected by moviepy, except on Windows! Windows users, before installing moviepy by hand, need to edit `moviepy/config_defaults.py` to provide the path to the imageMagick binary, which is called convert.

librosa:

librosa is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.

The first step of the program:

filename = librosa.util.example_audio_file()

gets the path to the audio example file included with librosa. After this step, `filename` will be a string variable containing the path to the example audio file. The example is encoded in OGG Vorbis format, so you will need the appropriate codec installed for audioread.

The second step:

y, sr = librosa.load(filename)

loads and decodes the audio as a time series `y`, represented as a one-dimensional NumPy floating point array. The variable `sr` contains the sampling rate of `y`, that is, the number of samples per second of audio. By default, all audio is mixed to mono and resampled to 22050 Hz at load time. This behavior can be overridden by supplying additional arguments to `librosa.load()`.

Next, we run the beat tracker:

tempo, beat_frames = librosa.beat.beat_track(y=y, sr=sr)

The output of the beat tracker is an estimate of the tempo (in beats per minute), and an array of frame numbers corresponding to detected beat events.

Frames here correspond to short windows of the signal (`y`), each separated by `hop_length = 512` samples. Since v0.3, *librosa* uses centered frames, so that the *k*th frame is centered around sample `k * hop_length`.

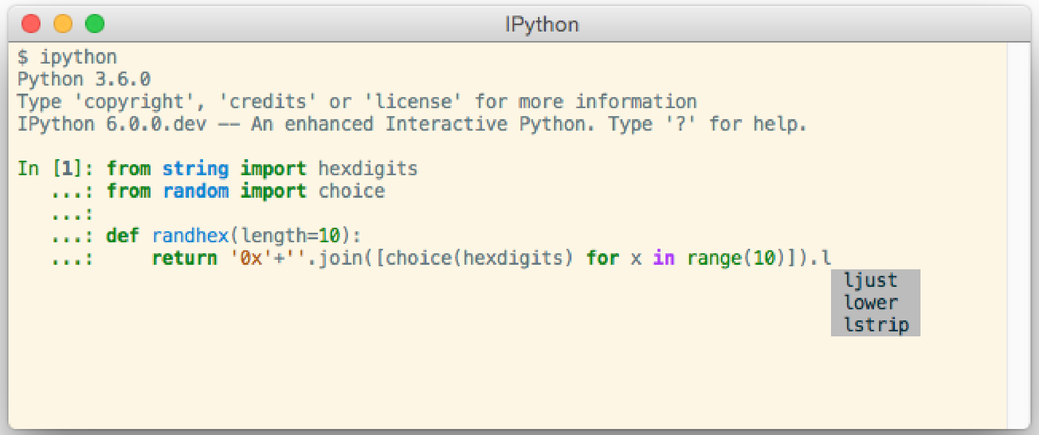
The next operation converts the frame numbers `beat_frames` into timings:

beat_times = librosa.frames_to_time(beat_frames, sr=sr)

Now, `beat_times` will be an array of timestamps (in seconds) corresponding to detected beat events.

ipython:

ipython provides a rich toolkit to help you make the most of using Python interactively.



```
$ ipython
Python 3.6.0
Type 'copyright', 'credits' or 'license' for more information
IPython 6.0.0.dev -- An enhanced Interactive Python. Type '?' for help.

In [1]: from string import hexdigits
   ...: from random import choice
   ...:
   ...: def randhex(length=10):
   ...:     return '0x'+''.join([choice(hexdigits) for x in range(10)]).l
                                                                      ljust
                                                                      lower
                                                                      lstrip
```

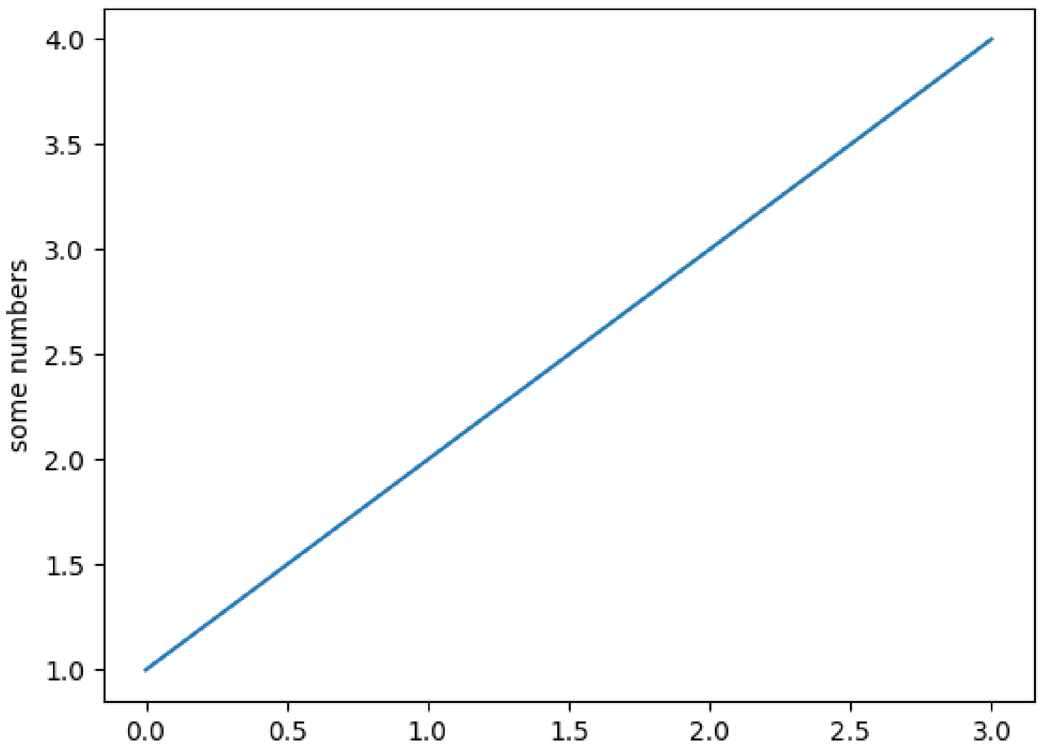
The Command line interface inherits the above functionality and adds

- real multi-line editing thanks to prompt toolkit.
- syntax highlighting as you type
- integration with command line editor for a better workflow.

matplotlib.pyplot:

matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

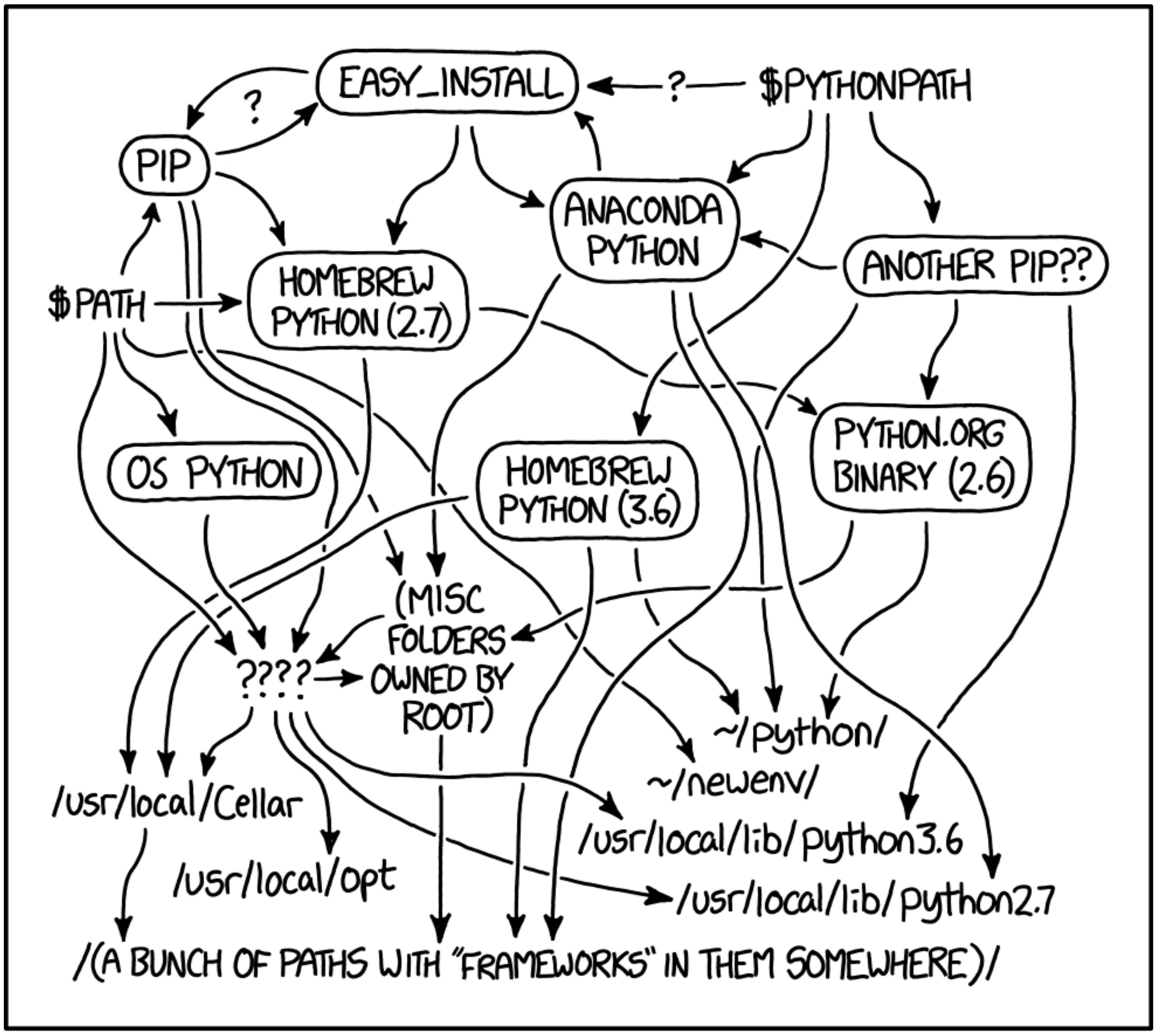
In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the *axes* part of a figure and not the strict mathematical term for more than one axis).



Simple Graph using matplotlib

numpy:

The only prerequisite for NumPy is Python itself. If you don’t have Python yet and want the simplest way to get started, we recommend you use the Anaconda Distribution - it includes Python, NumPy, and other commonly used packages for scientific computing and data science.



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

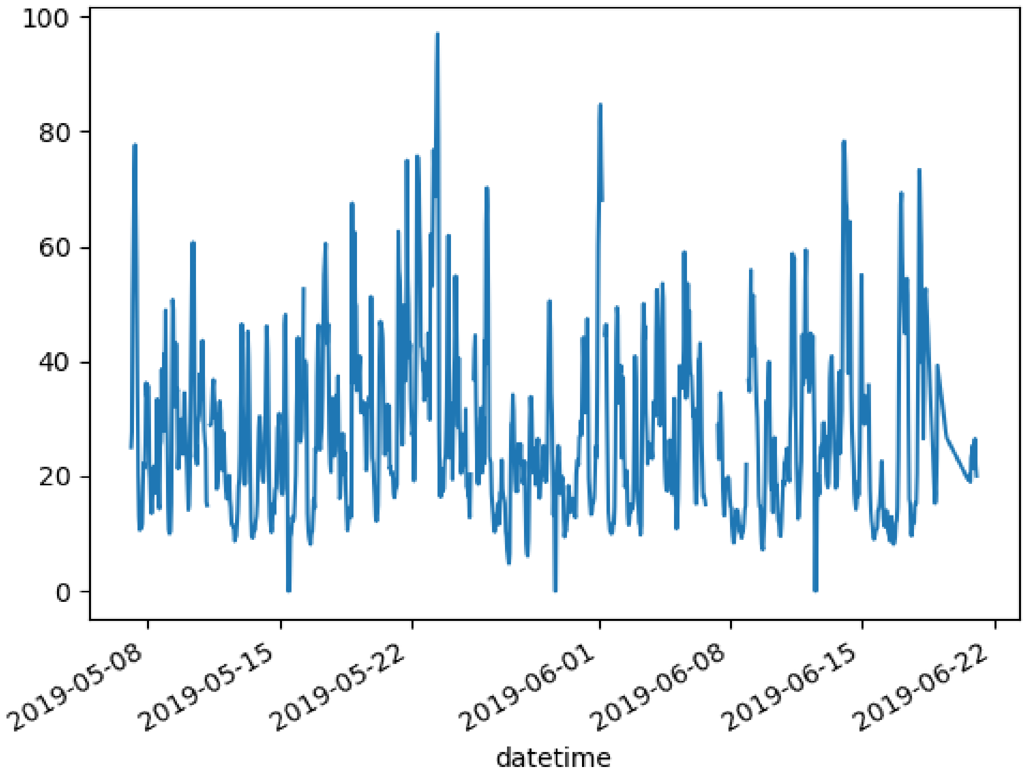
pandas:

To load the pandas package and start working with it, import the package. The community agreed alias for pandas is pd, so loading pandas as pd is assumed standard practice for all of the pandas documentation.

To manually store data in a table, create a DataFrame. When using a Python dictionary of lists, the dictionary keys will be used as column headers and the values in each list as rows of the DataFrame.

A **DataFrame** is a 2-dimensional data structure that can store data of different types (including characters, integers, floating point values, categorical data and more) in columns. It is similar to a spreadsheet, a SQL table or the data.frame in R.

The table has 3 columns, each of them with a column label. The column labels are respectively Name, Age and Sex. The column Name consists of textual data with each value a string, the column Age are numbers and the column Sex is textual data.



6. IMPLEMENTATION AND TESTING

Input video:

Getting YouTube URL:

```
import youtube_dl


ydl_opts = {}


def dwl_vid():
    with youtube_dl.YoutubeDL(ydl_opts) as ydl:
        ydl.download([zxt])


channel = 1
while (channel == int (1)):
    link_of_the_video = input ("Copy & paste the URL of the YouTube video you want to download: - ")
    zxt = link_of_the_video.strip()


    dwl_vid()
    channel = int(input("Enter 1 if you want to download more videos \n Enter 0 if you are done "))
```

Converting audio from input video:

```
# Python code to convert video to audio

import moviepy.editor as mp


# Insert Local Video File Path

clip = mp.VideoFileClip(r"india.mp4")


# Insert Local Audio File Path

clip.audio.write_audiofile(r"file.wav")
```

Breaking audio into chunks:

```
path='N:/major_project/newp/'

filename='file.wav'

import librosa

x, sr = librosa.load(path+filename,sr=16000)


int(librosa.get_duration(x, sr)/60)


max_slice=5

window_length = max_slice * sr


import IPython.display as ipd

a=x[21*window_length:22*window_length]

ipd.Audio(a, rate=sr)


energy = sum(abs(a**2))
```

Compute short time energy of every chunk:

```
energy = sum(abs(a**2))



import matplotlib.pyplot as plt

fig = plt.figure(figsize=(14, 8))

ax1 = fig.add_subplot(211)

ax1.set_xlabel('time')

ax1.set_ylabel('Amplitude')

ax1.plot(a)



import numpy as np

energy = np.array([sum(abs(x[i:i+window_length]**2)) for i in range(0, len(x),
window_lengt



import matplotlib.pyplot as plt

plt.hist(energy)

plt.show()
```

Compute every chunk as excitement or not:

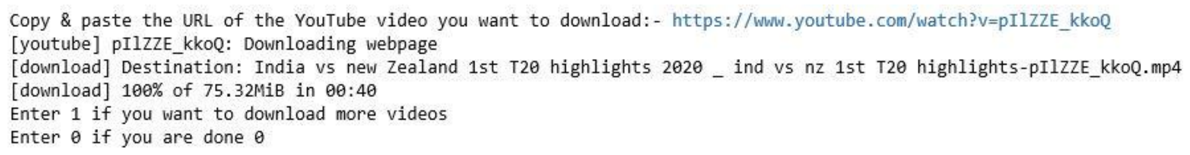
```
import pandas as pd

df=pd.DataFrame(columns=['energy','start','end'])


thresh=600

row_index=0

for i in range(len(energy)):

    value=energy[i]

    if(value>=thresh):

        i=np.where(energy == value)[0]

        df.loc[row_index,'energy']=value

        df.loc[row_index,'start']=i[0] * 5

        df.loc[row_index,'end']=(i[0]+1) * 5

        row_index= row_index + 1

temp=[]

i=0

j=0

n=len(df) - 2

m=len(df) - 1

while(i<=n):

  j=i+1

while(j<=m):
```

```
if(df['end'][i] == df['start'][j]):
   df.loc[i,'end'] = df.loc[j,'end']
   temp.append(j)
   j=j+1
  else:
   i=j
   break
df.drop(temp,axis=0,inplace=True)
```

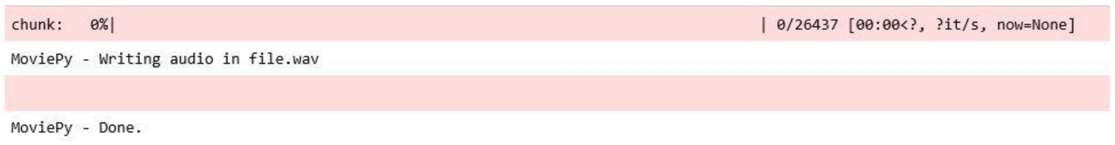
Merge all excitement clips:

```
from moviepy.video.io.ffmpeg_tools import ffmpeg_extract_subclip

start=np.array(df['start'])

end=np.array(df['end'])

filelist=list()

for i in range(len(df)):

    if(i!=0):

        start_lim = start[i] - 5

    else:

        start_lim = start[i]

    end_lim   = end[i]

    filename="N:/major_project/newp/c/" + str(i+1) + ".mp4"

    filelist.append(filename)

    ffmpeg_extract_subclip(path+"india.mp4",start_lim,end_lim,targetname=file

from moviepy.editor import VideoFileClip, concatenate_videoclips

clip=list()

for i in filelist:

clip1 = VideoFileClip(i)
   clip.append(clip1)

#clip2 = VideoFileClip("myvideo2.mp4").subclip(50,60)

#clip3 = VideoFileClip(filelist[1])

final_clip = concatenate_videoclips(clip)

final_clip.write_videofile("my_concat.mp4")
```

7. OUTPUTS

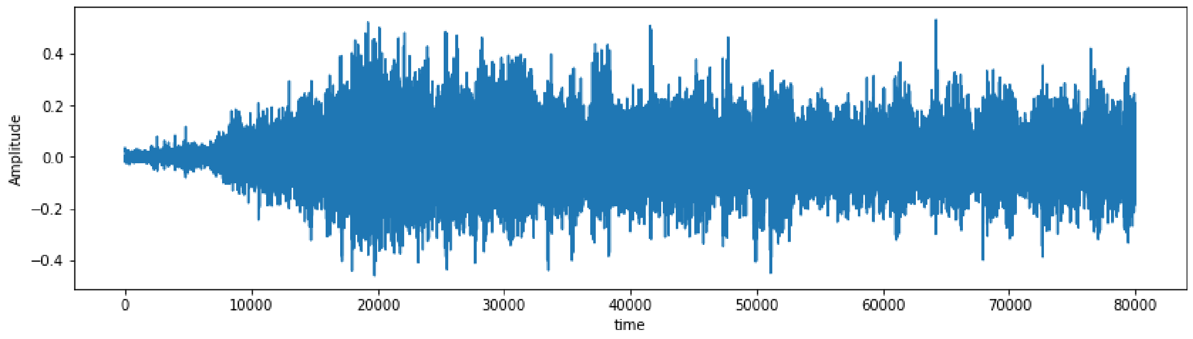


```
Copy & paste the URL of the YouTube video you want to download:- https://www.youtube.com/watch?v=pIlZZE_kkoQ
[youtube] pIlZZE_kkoQ: Downloading webpage
[download] Destination: India vs new Zealand 1st T20 highlights 2020 _ ind vs nz 1st T20 highlights-pIlZZE_kkoQ.mp4
[download] 100% of 75.32MiB in 00:40
Enter 1 if you want to download more videos
Enter 0 if you are done 0
```

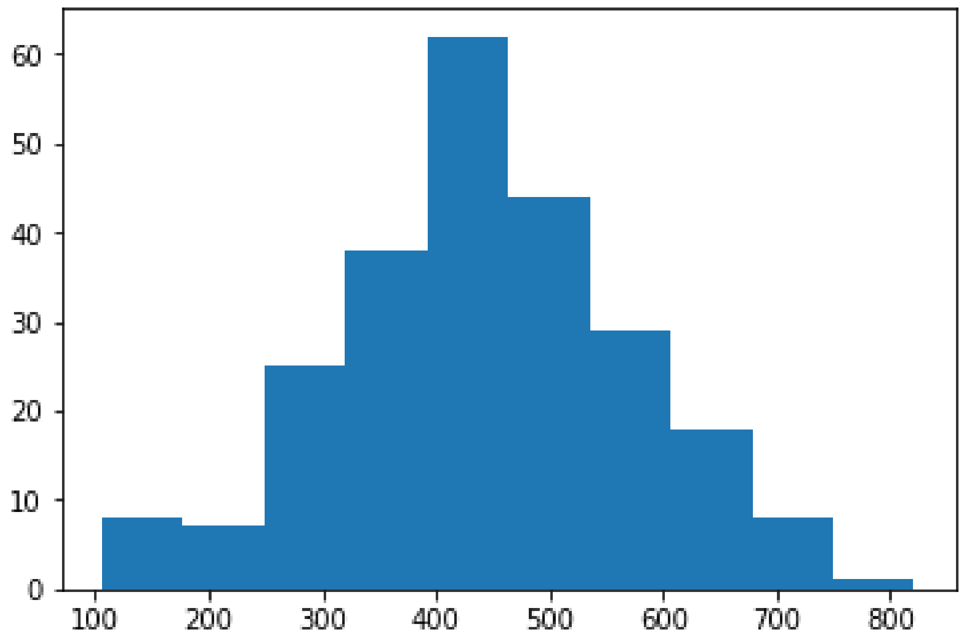


```
chunk:   0%|                                                                    | 0/26437 [00:00<?, ?it/s, now=None]
MoviePy - Writing audio in file.wav

MoviePy - Done.
```

Time-series domain



Energy Distribution Graph



Final Highlight Video

7. CONCLUSION AND FORESEEABLE ENHANCEMENT

In this Project we conclude that using librosa package it is easier to automatically generate highlights using python. During this work, librosa libraries have been studied in depth. Input is directly taken from you tube.

In future, the work can be extended and improved for other sports also rather than cricket. We can also implement other approaches like computer vision approach for generating highlights in future.

BIBLIOGRAPHY

1. Aravind Pal (2019) Become a video analysis expert: A Simple Approach to Automatically Generating Highlights using Python, blog: Analytics Vidhya.
2. https://librosa.github.io/librosa/
3. https://pypi.org/project/moviepy/
4. https://ipython.org/
5. https://matplotlib.org/
6. https://matplotlib.org/api/pyplot_api.html
7. https://numpy.org/
8. https://pandas.pydata.org/