COVER PAGE

CERTIFICATE

ACKNOWLEDGEMENT

INDEX

|  |  |  |
| --- | --- | --- |
| **Sr.no** | **Topic** | **Page no** |
|  | Aim | 1 |
|  | Objectives and Scope of the Project | 2 |
|  | System Requirements | 3 |
|  | Salient Features | 4 |
|  | A Walk Through | 5 |
|  | Bibliography | 37 |

AIM

The Aim of the project is the allow the user to download You-Tube Videos with ease. The Project must also be able to satisfy the following objectives :

1. The User should be able to download You-Tube Videos in *1080p and 720p,* depending on the upload resolution of the video.
2. The User should be able to download the same video in a *lower* *resolution* of *360p*, or *480p* as well.
3. The User should also be able to download *only the sound* of the video, as well as only the Video in it’s *Highest* *Resolution*.
4. The User should be able to *Change the path of the downloaded video* in the computer within the software using a File selection Dialog Box
5. The User must be able to view the *progress* of the video downloaded, both as a *Progress Bar*, and also in *Percentages*.
6. The Software should show an *Estimated* *File* *Size* for each resolution of the video downloaded.
7. The User Must also be able to download *entire You-Tube playlists*, deselect, *Select,* and *remove* certain videos from the playlist, and must be able to see the *count* of the *videos* *Downloaded* and *Remaining*.
8. The User Must be able to view *Graphs*, and *Statistics* regarding the Videos that have been downloaded by the software.
9. The Data of all the downloaded videos must be saved as a *‘.csv’* file, and processed by *Pandas* in the form of a *DataFrame*.

OBJECTIVE AND SCOPE OF THE PROJECT

* Scope of the Project :

1. Download Quality :
2. 1080p
3. 720p
4. 480p
5. 2160p
6. 120 Hz audio
7. Download Format :
8. .mp4
9. .webm
10. .ogg
11. .wav

* Limitations of the Project

1. Can only download video and linked audio up to 720p
2. Some Playlist links are not supported, ones that are updated by youtube and protected under copyright laws.
3. Only a maximum of 100 videos can be download in a single playlist at once.
4. Requires constant maintenance, and version updates.
5. Heavy Reliability on foreign Dependencies and modules.
6. Does not support multiple downloads at once.
7. Does not make efficient use of the network bandwidth.
8. Graphs do not get updated immediately after updating the csv file.
9. Cannot Resume downloading videos from the point of interruption.

SYSTEM REQUIREMENTS

* Dependencies and System Requirements :

1. *Active Connection* to internet *while running* the program
2. Minimum of a *x64* or *x86* computer
3. *100* *MB* *RAM*, *200* *MB* Free Storage Space
4. Monitor Resolution – *1280* x *720p*
5. *Windows 7, 8, or 10.*

* Compilation of Source Code

1. *Python 3.7 to 3.9* as of date of writing.
2. Modules *pytube* 10.0, *pillow*, *requests*, *pandas*, *matplotlib*, *youtube*-*dl*
3. Run the following Commands in the *command prompt* of an *activated* virtual environment of the suggested python version, or the via pip *installed in the path* of the computer to download the required library dependencies.

pip install pytube

pip install pillow

pip install requests

pip install pandas

pip install matplotlib

pip install youtube-dl

SALIENT FEATURES

* 1. Can download *Any Youtube Video* from the URL
  2. Can download the video in either *720p* with *360p* with *Audio*
  3. Can download *only the audio* from a video.
  4. Can download *multiple selected or consecutive videos* from a *Playlist* URL
  5. Can save the video in *any path* of the Computer
  6. Can show the *progress* of the Video that is being downloaded.
  7. Can show the *File size prior to Downloading.*
  8. Has a *graphical, intuitive, and easy to use User Interface and Experience*
  9. Can show *data* and *statistics* of the video that you have downloaded.
  10. Project is *segregated*, *commented*, and *organized* into *functions*, *classes*, and *files* making it easy to read.
  11. Uses *Pandas Data Frames* to manage the statistics of the user.
  12. Can show *graphs* of the *data* Collected from the user based on the Videos *downloaded*.
  13. Uses *matplotlib* to show the *graphs* to the user.
  14. Uses *tkinter* for the *created GUI* and *python's file management* to store the user data.
  15. *Offers* *Portability* to the user in the form of a *single .exe* file to download and use.
  16. Uses *CPU threading* for doing longer tasks more efficiently, thereby making the program run faster.
  17. Uses *CSV Files* for Data Storage

A WALK THROUGH

Working of the Pytube Library:

Every youtube video has a URL that is unique to the video.

Example :

https://www.youtube.com/watch?v=8kooIgKESYE

This video has a common part https://www.youtube.com/watch?v= and a unique part 8kooIgKESYE

The string 8kooIgKESYE is the identification for the video, and is the address where the video is stored in the server. It is a base 64 number.

To uniquely store the thumbnails of the videos that we download in the project, we use this number or string as the name of the thumbnail, as can be seen in the script.

Because not everyone has the same internet speed, when you upload a video, youtube stores the video in many formats ranging from 144p to the highest resolution of the uploaded video

If the video is greater than 1080p in resolution, it is difficult for a 4G network to download and render the video on the screens of the users, and hence it is broken into audio and video. These different versions of the video are called streams. Pytube has a Youtube and a Playlist class. So, to download a video, we create an instance of the Class, an object, that has the URLs of all these streams.

To download the video, the user selects the quality, and we give that quality of stream to pytube's youtube object, after which we can download the video.

Downloading the Playlists

`https://www.youtube.com/watch?v=-GhzpvvIXlM&list=PLS1QulWo1RIY6fmY\_iTjEhCMsdtAjgbZM`

is an example playlist, and follows much of the same rules as the other URL. A playlist is just

a bunch of video URL's that are given a name, an author and creation time.

This data (the name, video URLs etc. are stored on the server, and that location is provided to us by the playlist URL) so, with this URL, we can simply get the Youtube video URLs, and then the process of creation of a youtube object is repeated and the video is downloaded.

Graphing and Storing the Data

From the Youtube object, you can get data about the youtube video such as

* 1. title
  2. views
  3. rating
  4. author
  5. description
  6. Date of Publication
  7. Likes to dislike ratio
  8. Like count and dislike count

so, after downloading a video, this data is written to a single file, stored in `data/video\_data.csv`

This data is written such that it does not overlap, and there are no repeated entries Storing this data in csv is that it can be retrieved later anytime.

We can retrieve the data from these files using the `fileIO.py` file. This file has a `read` class, that has functions like `get\_title()` or `get\_views()` etc. These functions can be used to get a list that has all the titles or the authors or the views of all the videos that we downloaded. The first element of each file corresponds to the first video downloaded.

These lists can then be further used to create pandas series, dataframes, and then create graphs. These functions will be written in the `graphs.py` file.

These graphs are then integrated in the `main.py` file that are then displayed as statistics in the program.

Code

# importing the things

# import os

import os

import tkinter as tk

import time

from tkinter import ttk

import pytube as pt

import requests

from tkinter.filedialog import askopenfilename

import youtube\_dl

from PIL import ImageTk, Image

import threading

import Tags

import re

import File\_IO as fio

import methods

import Graphs

# defining Some constants

HEIGHT = 720

WIDTH = 1280

TYPE = 'SINGLE'

# INTRO\_BGIMG = "C:/Users/littl/Desktop/Programs/Python/Kappa Video Downloader/Assets/Background Images/INTRO BGIMG.png"

INTRO\_BGIMG = "Assets/Background Images/INTRO BGIMG.png"

VIDDOWN\_SINGLE\_BGIMG = "Assets/Background Images/VIDDOWN SINGLE BGIMG 3.png"

VIDDOWN\_MULTIPLE\_BGIMG = "Assets/Background Images/VIDDOWN MULTIPLE BGIMG.png"

DOWNLOAD\_IMAGE = "Assets/Background Images/downloadbtn.png"

REMOVE\_IMAGE = "Assets/Background Images/remove.png"

FILE\_SELECT\_IMAGE = "Assets/Background Images/FILE SELECT1.png"

RESTART\_IMAGE = 'Assets/Background Images/onemore.png'

PROCEED\_BTN = 'Assets/Background Images/PROCEED BTN.png'

STATISTICS\_BTN = 'Assets/Background Images/STATISTICS BTN.png'

URL = ''

FILENAME = os.getcwd()

maxbytes = 0

sth = 1

again = True

done = False

video\_titles = []

video\_titles\_with\_urls = []

like\_counts = []

dislike\_counts = []

Quality\_tag = 142

filesize = 0

playlist\_URLS = [ ]

download\_list = [ ]

sel\_stream = '360p'

'''Class that has functions for displaying windows and doing all the work in the program'''

class window :

@staticmethod

def intro\_win() :

"""

function that displays the introduction tkinter window, and has enter button that closes it.

Also checks if the link is a playlist or a single video, and if it is, then it changes the

type from single to playlist it then returns the url of the video

"""

global TYPE

# checks if the url is valid, changes the global volues.

def proceed() :

global TYPE, URL

user\_url = entry.get()

r = requests.get( user\_url ) # random video id

if "Video unavailable" in r.text :

print( 'video is invalid' )

TYPE = 'invalid'

else :

if 'list=' in user\_url :

TYPE = 'PLAYLIST'

print( 'its a playlist' )

else :

TYPE = 'SINGLE'

print( 'this is the url', user\_url )

URL = user\_url

# runs if you pressed the statistics button, redirects you to the statistics page.

def statistics() :

global TYPE

TYPE = 'STATISTICS'

# Beginning loop from here

root = tk.Tk()

canvas = tk.Canvas( root, height = HEIGHT, width = WIDTH )

canvas.pack()

# Placing the background image in the canvas

BG\_IMG = tk.PhotoImage( file = INTRO\_BGIMG, master = root )

BG\_IMG\_LABEL = tk.Label( canvas, image = BG\_IMG )

BG\_IMG\_LABEL.place( relwidth = 1, relheight = 1 )

# placing the entry box for entering the url

entry = tk.Entry( canvas, bg = 'white', font = ("Calibre", 13) )

entry.place( rely = 0.80, relx = 0.24, relwidth = 0.55, relheight = 0.05 )

proceed\_img = Image.open( PROCEED\_BTN )

proceed\_img = proceed\_img.resize( (141, 43), Image.ANTIALIAS )

proceed\_img = ImageTk.PhotoImage( proceed\_img )

statistics\_img = Image.open( STATISTICS\_BTN )

statistics\_img = statistics\_img.resize( (141, 43), Image.ANTIALIAS )

statistics\_img = ImageTk.PhotoImage( statistics\_img )

# placing the proceed button, that calls the proceed function

proceed\_btn = tk.Button( canvas, image = proceed\_img, command = lambda : [ proceed(), root.destroy() ],

bg = '#64A8E8', border = 0, activebackground = '#64A8E8' )

proceed\_btn.place( rely = 0.9, relx = 0.38 )

# placing the Statistics button, that calls the Statistics window

statistics\_btn = tk.Button( canvas, image = statistics\_img, command = lambda : [ statistics(), root.destroy() ],

bg = '#64A8E8', border = 0, activebackground = '#64A8E8' )

statistics\_btn.place( rely = 0.9, relx = 0.51 )

root.mainloop()

# This is the backup

@staticmethod # this thing works tho...

def sel\_download\_win\_single( url, video\_obj ) :

"""

this window shows you the thumbnail of the video along with its title and available qualities, also shows you the

download button and the file path selection menu. You click download and the video downloades.

"""

tnurl = video\_obj.thumbnail\_url

length = video\_obj.length

# author = video\_obj.author

title = video\_obj.title

global again, sel\_stream

# on change dropdown value, and link to the main menu

def change\_dropdown( \*args ) :

global sel\_stream

sel\_stream = tkvar.get()

print( 'value of the sel stream is : ', sel\_stream )

video\_type = video\_obj.streams.get\_by\_itag(

list( Tags.tags.keys() )[ list( Tags.tags.values() ).index( sel\_stream ) ] )

mbytes = (round( video\_type.filesize / 1000000, 2 )).\_\_str\_\_() + ' MB'

print( mbytes )

file\_size\_lbl.config( text = mbytes.\_\_str\_\_() )

# opens the file explorer window to select the folder to download, and changes the global file path variable

def open\_file\_explorer() :

global FILENAME

tk.Tk().withdraw()

FILENAME = tk.filedialog.askdirectory()

print( FILENAME )

file\_path.config( text = FILENAME )

# to show the progess bar, and update the values of the percentage downloaded

def on\_progress\_dothis( stream, chunk: bytes, bytes\_remaining: int ) -> None : # pylint: disable=W0613

Bytes = maxbytes - bytes\_remaining

percent = round( (100 \* (maxbytes - bytes\_remaining)) / maxbytes, 2 )

downloading = percent.\_\_str\_\_() + '%'

progress\_bar[ "value" ] = Bytes

progress\_value.config( text = downloading )

root.update\_idletasks()

if percent == 100.0 :

downloading = 'Done! '

progress\_value.config( text = downloading )

# to download the video, part of the threading process, then calls the on\_progress\_do\_this() function

def download() :

global maxbytes

print( "Accessing YouTube URL..." )

video = pt.YouTube( url, on\_progress\_callback = on\_progress\_dothis )

video\_type = video.streams.get\_by\_itag(

list( Tags.tags.keys() )[ list( Tags.tags.values() ).index( sel\_stream ) ] )

print( "Fetching" )

maxbytes = video\_type.filesize

mbytes = (round( video\_type.filesize / 1000000, 2 )).\_\_str\_\_() + ' MB'

print( mbytes )

file\_size\_lbl.config( text = mbytes.\_\_str\_\_() )

progress\_bar[ "maximum" ] = maxbytes

print( maxbytes )

video\_type.download( FILENAME )

# quits the window, after changing some global variables

def restart() :

global again

again = True

root.destroy()

pass

# Starting the loop

root = tk.Tk()

tkvar = tk.StringVar( root )

print( tkvar )

# Defining some image variables to be used in the buttons and the thumbnails

dimg = Image.open( DOWNLOAD\_IMAGE )

dimg = dimg.resize( (167, 51), Image.ANTIALIAS )

dimg = ImageTk.PhotoImage( dimg )

flsimg = Image.open( FILE\_SELECT\_IMAGE )

flsimg = flsimg.resize( (78, 51), Image.ANTIALIAS )

flsimg = ImageTk.PhotoImage( flsimg )

dnimg = Image.open( RESTART\_IMAGE )

dnimg = dnimg.resize( (125, 125), Image.ANTIALIAS )

dnimg = ImageTk.PhotoImage( dnimg )

BG\_IMG = tk.PhotoImage( file = VIDDOWN\_SINGLE\_BGIMG )

# Creating the Canvas

canvas = tk.Canvas( root, height = HEIGHT, width = WIDTH )

canvas.pack()

# Placing the background image in the canvas

BG\_IMG\_LABEL = tk.Label( canvas, image = BG\_IMG )

BG\_IMG\_LABEL.place( relwidth = 1, relheight = 1 )

# scrapping the thumbnail from the current video and putting it in some folder

methods.get\_video\_tnl( url, tnurl )

img = Image.open( os.path.join( 'Assets/Thumbnails', methods.get\_vid\_id( url ) + '.png' ) )

img = img.resize( (283, 160), Image.ANTIALIAS )

img = ImageTk.PhotoImage( img )

# displaying the thumbnail

video\_tnl = tk.Label( canvas, image = img )

video\_tnl.place( relx = 0.01, rely = 0.2 )

# displaying the title of the video

vid\_title = tk.Label( canvas, text = title, anchor = 'w', font = (

"Calibre", 18), bg = 'white', wraplength = 800 )

vid\_title.place( rely = 0.2, relx = 0.25 )

# displaying the length of the video

vid\_len = methods.conv\_len( length )

vid\_length = tk.Label( canvas, text = vid\_len, anchor = 'w', font = (

"Calibre", 18), bg = 'white', wraplength = 400 )

vid\_length.place( rely = 0.38, relx = 0.25 )

# Creating the drop down menu

qualities = Tags.get\_available\_qualities\_with\_obj( video\_obj )

tkvar.set( qualities[ 0 ] ) # set the default option

popupMenu = tk.OptionMenu( canvas, tkvar, \*qualities )

popupMenu.place( relx = 0.3, rely = 0.52, relwidth = 0.2, relheight = 0.05 )

tkvar.trace( 'w', change\_dropdown )

# Displaying the download button

down\_btn = tk.Button( canvas, image = dimg, command = lambda : threading.Thread( target = download ).start(),

font = ("Calibre", 16), bg = 'white', border = 0, activebackground = 'white' )

down\_btn.place( rely = 0.9, relx = 0.85 )

# displaying the file selection button

file\_selection\_btn = tk.Button( canvas, image = flsimg, command = open\_file\_explorer, font = (

"Calibre", 16), bg = 'white', border = 0, activebackground = 'white' )

file\_selection\_btn.place( rely = 0.6, relx = 0.25 )

# displaying the file path text box

file\_path = tk.Label( canvas, text = FILENAME, font = ("Calibre", 18, 'italic'), bg = 'white', )

file\_path.place( rely = 0.68, relx = 0.25 )

progress\_bar = ttk.Progressbar( canvas, orient = "horizontal", length = 200, mode = "determinate" )

progress\_bar.place( rely = 0.82, relx = 0.05, relwidth = 0.7, relheight = 0.05 )

progress\_bar[ 'value' ] = 0

# displaying the amount of video downlaoded

progress\_value = tk.Label( canvas, text = '', font = ("Calibre", 18), bg = 'white' )

progress\_value.place( rely = 0.895, relx = 0.25 )

# displaying the file size

file\_size\_lbl = tk.Label( canvas, text = "0 MB", font = ("Calibre", 19), bg = 'white' )

file\_size\_lbl.place( rely = 0.525, relx = 0.8 )

# displaying the button for downloading another video, that is restarting the program

next\_btn = tk.Button( canvas, image = dnimg, command = restart, font = ("Calibre", 16), bg = '#8CB0FF', border = 0,

activebackground = '#8CB0FF' )

next\_btn.place( rely = 0.01, relx = 0.9 )

root.mainloop()

@staticmethod

def sel\_downlaod\_win\_playlist( playlist\_obj ) :

global download\_list

"""

this window shows you the thumbnail of the video along with its title and available qualities, also shows you the

download button and the file path selection menu. You click download and the video downloades.

"""

download\_list = playlist\_obj.video\_urls

video\_obj = pt.YouTube(download\_list[0])

total\_vids = len( playlist\_obj.video\_urls )

tnurl = video\_obj.thumbnail\_url

cur\_video\_length = video\_obj.length

cur\_video\_title = video\_obj.title

global again, sel\_stream, filesize

# on change dropdown value, and link to the main menu

def change\_dropdown( \*args ) :

global sel\_stream

sel\_stream = tkvar.get()

print( 'value of the sel stream is : ', sel\_stream )

video\_type = video\_obj.streams.get\_by\_itag(

list( Tags.tags.keys() )[ list( Tags.tags.values() ).index( sel\_stream ) ] )

mbytes = (round( video\_type.filesize / 1000000, 2 )).\_\_str\_\_() + ' MB'

print( mbytes )

file\_size\_lbl.config( text = mbytes.\_\_str\_\_() )

# opens the file explorer window to select the folder to download, and changes the global file path variable

def open\_file\_explorer() :

global FILENAME

tk.Tk().withdraw()

FILENAME = tk.filedialog.askdirectory()

print( FILENAME )

file\_path.config( text = FILENAME )

# to show the progess bar, and update the values of the percentage downloaded

def on\_progress\_dothis( stream, chunk: bytes, bytes\_remaining: int ) -> None : # pylint: disable=W0613

Bytes = maxbytes - bytes\_remaining

percent = round( (100 \* (maxbytes - bytes\_remaining)) / maxbytes, 2 )

downloading = percent.\_\_str\_\_() + '%'

progress\_bar[ "value" ] = Bytes

progress\_value.config( text = downloading )

root.update\_idletasks()

if percent == 100.0 :

downloading = 'Done! '

progress\_value.config( text = downloading )

# to download the video, part of the threading process, then calls the on\_progress\_do\_this() function

def download() :

global maxbytes, total\_vids

total\_vids = len( download\_list )

downloaded = 0

skipped = 0

print( len( download\_list ) )

for vids in download\_list :

print( "Accessing YouTube URL..." )

vid = pt.YouTube( vids, on\_progress\_callback = on\_progress\_dothis )

video\_type = vid.streams.get\_by\_itag(

list( Tags.tags.keys() )[ list( Tags.tags.values() ).index( sel\_stream ) ] )

methods.get\_video\_tnl( vid.watch\_url, vid.thumbnail\_url )

img1 = Image.open( os.path.join( 'Assets/Thumbnails',

methods.get\_vid\_id( vid.watch\_url ) + '.png' ) )

img1 = img1.resize( (283, 160), Image.ANTIALIAS )

img1 = ImageTk.PhotoImage( img1 )

cur\_vid\_length = vid.length

vid\_len1 = methods.conv\_len( cur\_vid\_length )

cur\_vid\_title = vid.title

vid\_title.config( text = cur\_vid\_title )

vid\_length.config( text = vid\_len1 )

video\_tnl.config( image = img1 )

print( "Fetching" )

maxbytes = video\_type.filesize

mbytes = (round( video\_type.filesize / 1000000, 2 )).\_\_str\_\_() + ' MB'

print( mbytes )

file\_size\_lbl.config( text = mbytes.\_\_str\_\_() )

progress\_bar[ "maximum" ] = maxbytes

print( maxbytes )

video\_type.download( FILENAME )

downloaded += 1

downloaded\_lbl.config( text = downloaded.\_\_str\_\_() )

remaining = total\_vids - downloaded

remaining\_lbl.config( text = remaining )

skipped\_lbl.config( text = skipped )

# quits the window, after changing some global variables

def restart() :

global again

again = True

root.destroy()

pass

def remove() :

"""used to remove the selected things from the menu of showing videos"""

global download\_list

print( 'you clicked remove' )

for item in reversed( all\_videos.curselection() ) :

all\_videos.delete( item )

download\_list = [ ]

download\_list = all\_videos.get( 0, "end" )

new\_list = []

for i in range(len(download\_list)):

new\_list.append(download\_list[i][1])

download\_list = new\_list

remaining\_lbl.config( text = len( download\_list ) )

total\_vids\_lbl.config( text = len( download\_list ) )

# Starting the loop

root = tk.Tk()

tkvar = tk.StringVar( root )

print( tkvar )

# Defining some image variables to be used in the buttons and the thumbnails

dimg = Image.open( DOWNLOAD\_IMAGE )

dimg = dimg.resize( (167, 51), Image.ANTIALIAS )

dimg = ImageTk.PhotoImage( dimg )

rimg = Image.open( REMOVE\_IMAGE )

rimg = rimg.resize( (170, 30), Image.ANTIALIAS )

rimg = ImageTk.PhotoImage( rimg )

flsimg = Image.open( FILE\_SELECT\_IMAGE )

flsimg = flsimg.resize( (60, 40), Image.ANTIALIAS )

flsimg = ImageTk.PhotoImage( flsimg )

dnimg = Image.open( RESTART\_IMAGE )

dnimg = dnimg.resize( (125, 125), Image.ANTIALIAS )

dnimg = ImageTk.PhotoImage( dnimg )

BG\_IMG = tk.PhotoImage( file = VIDDOWN\_MULTIPLE\_BGIMG )

# Creating the Canvas

canvas = tk.Canvas( root, height = HEIGHT, width = WIDTH )

canvas.pack()

# Placing the background image in the canvas

BG\_IMG\_LABEL = tk.Label( canvas, image = BG\_IMG )

BG\_IMG\_LABEL.place( relwidth = 1, relheight = 1 )

# scrapping the thumbnail from the current video and putting it in some folder

methods.get\_video\_tnl( video\_obj.watch\_url, tnurl )

# creating the image object for the thumbnails

img = Image.open( os.path.join( 'Assets/Thumbnails',

methods.get\_vid\_id( video\_obj.watch\_url ) + '.png' ) )

img = img.resize( (283, 160), Image.ANTIALIAS )

img = ImageTk.PhotoImage( img )

# displaying the thumbnail

video\_tnl = tk.Label( canvas, image = img )

video\_tnl.place( relx = 0.01, rely = 0.2 )

# displaying the title of the video

vid\_title = tk.Label( canvas, text = cur\_video\_title, anchor = 'w',

font = ("Calibre", 18), bg = 'white', wraplength = 800 )

vid\_title.place( rely = 0.2, relx = 0.25 )

# displaying the length of the video

vid\_len = methods.conv\_len( cur\_video\_length )

vid\_length = tk.Label( canvas, text = vid\_len, anchor = 'w', font = (

"Calibre", 18), bg = 'white', wraplength = 400 )

vid\_length.place( rely = 0.38, relx = 0.25 )

# Creating the drop down menu

qualities = Tags.get\_available\_qualities\_with\_obj( video\_obj )

tkvar.set( qualities[ 0 ] ) # set the default option

popupMenu = tk.OptionMenu( canvas, tkvar, \*qualities )

popupMenu.place( relx = 0.55, rely = 0.395, relheight = 0.05 )

tkvar.trace( 'w', change\_dropdown )

# Displaying the download button

down\_btn = tk.Button( canvas, image = dimg, command = lambda : threading.Thread(

target = download ).start(), font = ("Calibre", 16), bg = 'white', border = 0, activebackground = 'white' )

down\_btn.place( rely = 0.9, relx = 0.85 )

# displaying the file selection button

file\_selection\_btn = tk.Button( canvas, image = flsimg, command = open\_file\_explorer, font = (

"Calibre", 16), bg = 'white', border = 0, activebackground = 'white' )

file\_selection\_btn.place( rely = 0.38, relx = 0.92 )

remove\_btn = tk.Button( canvas, image = rimg, command = remove, font = (

"Calibre", 16), bg = 'white', border = 0, activebackground = 'white' )

remove\_btn.place( rely = 0.8, relx = 0.02 )

# displaying the file path text box

file\_path = tk.Label( canvas, text = FILENAME, font = ("Calibre", 18, 'italic'), bg = 'white', )

file\_path.place( rely = 0.85, relx = 0.10 )

# Displaying the scrollbar next to the thing

scrollbar = tk.Scrollbar( root )

scrollbar.place( rely = 0.53, relx = 0.67, relheight = 0.25 )

# displaying the listbox

all\_videos = tk.Listbox( canvas, yscrollcommand = scrollbar.set, width = 70, font = ("Calibre", 16, 'italic'), height = 7,

selectmode = tk.EXTENDED )

for video in video\_titles\_with\_urls :

all\_videos.insert( tk.END, video )

all\_videos.place( rely = 0.533, relx = 0.02 )

scrollbar.config( command = all\_videos.yview )

# displaying the progressbar from downlaoding the current video

progress\_bar = ttk.Progressbar( canvas, orient = "horizontal", length = 200, mode = "determinate" )

progress\_bar.place( rely = 0.915, relx = 0.15, relwidth = 0.6, relheight = 0.03 )

progress\_bar[ 'value' ] = 0

# displaying the amount of video downlaoded

progress\_value = tk.Label( canvas, text = '0 %', font = ("Calibre", 19), bg = 'white' )

progress\_value.place( rely = 0.91, relx = 0.76 )

# displaying the file size

file\_size\_lbl = tk.Label( canvas, text = "0 MB", font = ("Calibre", 19), bg = 'white' )

file\_size\_lbl.place( rely = 0.815, relx = 0.88 )

# displaying the number of videos that we skipped coz they were unavailable to download due to some reason or error

skipped\_lbl = tk.Label( canvas, text = "0", font = ("Calibre", 19), bg = 'white' )

skipped\_lbl.place( rely = 0.75, relx = 0.92 )

# displaying the remaining number of videos from the selected ones

remaining\_lbl = tk.Label( canvas, text = total\_vids, font = ("Calibre", 19), bg = 'white' )

remaining\_lbl.place( rely = 0.69, relx = 0.92 )

# displaying the number of videos that we finished downloading

downloaded\_lbl = tk.Label( canvas, text = "0", font = ("Calibre", 19), bg = 'white' )

downloaded\_lbl.place( rely = 0.63, relx = 0.92 )

# displaying whichth number of video it is that we are downloading from our selected list

cur\_number\_lbl = tk.Label( canvas, text = "0", font = ("Calibre", 19), bg = 'white' )

cur\_number\_lbl.place( rely = 0.57, relx = 0.92 )

# displaying the total number of videos in the playlist given by the user

total\_vids\_lbl = tk.Label( canvas, text = total\_vids, font = ("Calibre", 19), bg = 'white' )

total\_vids\_lbl.place( rely = 0.51, relx = 0.92 )

# displaying the button for downloading another video, that is restarting the program

next\_btn = tk.Button( canvas, image = dnimg, command = restart, font = ("Calibre", 16), bg = '#8CB0FF', border = 0,

activebackground = '#8CB0FF' )

next\_btn.place( rely = 0.01, relx = 0.9 )

root.mainloop()

@staticmethod

def statistics() :

root = tk.Tk()

root.geometry( "1280x720" )

# Creating the notebook, that enables tabs

my\_notebook = ttk.Notebook( root )

my\_notebook.pack()

# \_\_\_\_\_\_\_\_\_\_\_\_\_Views Tab\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

base\_frame\_tab\_1 = tk.Frame( my\_notebook, width = 1280, height = 5000, bg = "#65A8E8" )

# Adding a scrollbar to that tab

base\_canvas\_tab\_1 = tk.Canvas( base\_frame\_tab\_1, width = 1250, height = 5000 )

base\_canvas\_tab\_1.pack( side = tk.LEFT, fill = tk.BOTH, expand = 1 )

my\_scrollbar = ttk.Scrollbar( base\_frame\_tab\_1, orient = tk.VERTICAL, command = base\_canvas\_tab\_1.yview )

my\_scrollbar.pack( side = tk.RIGHT, fill = tk.Y )

base\_canvas\_tab\_1.configure( yscrollcommand = my\_scrollbar.set )

base\_canvas\_tab\_1.bind( '<Configure>', lambda e : base\_canvas\_tab\_1.configure( scrollregion = base\_canvas\_tab\_1.bbox( "all" ) ) )

tab\_frame\_1 = tk.Frame( base\_canvas\_tab\_1, width = 1280, height = 5000, bg = "#65A8E8" )

base\_canvas\_tab\_1.create\_window( (0, 0), window = tab\_frame\_1, anchor = "nw" )

# \_\_\_\_Stuff in the tab\_\_\_#

bg\_label\_1 = tk.Label( tab\_frame\_1, text = 'Views Vs Videos Downloaded', bg = '#65A8E8', font = ("Calibre", 30) )

bg\_label\_1.place( relx = 0.3, rely = 0.01 )

views\_graph\_img = ImageTk.PhotoImage( Image.open( 'Assets/Graphs/views\_line\_graph.png' ) )

views\_graph\_img\_lbl = tk.Label( tab\_frame\_1, image = views\_graph\_img )

views\_graph\_img\_lbl.place( relx = 0.25, rely = 0.03 )

views\_graph\_img\_2 = ImageTk.PhotoImage( Image.open( 'Assets/Graphs/views\_bar\_graph.png' ) )

views\_graph\_img\_2\_lbl = tk.Label( tab\_frame\_1, image = views\_graph\_img\_2 )

views\_graph\_img\_2\_lbl.place( relx = 0.25, rely = 0.13 )

views\_graph\_img\_3 = ImageTk.PhotoImage( Image.open( 'Assets/Graphs/views\_hist\_graph.png' ) )

views\_graph\_img\_3\_lbl = tk.Label( tab\_frame\_1, image = views\_graph\_img\_3 )

views\_graph\_img\_3\_lbl.place( relx = 0.25, rely = 0.23 )

# \_\_\_#

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ratings Tab\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

base\_frame\_tab\_2 = tk.Frame( my\_notebook, width = 1280, height = 5000, bg = "#65A8E8" )

# Adding scrollbar to that tab

base\_canvas\_tab\_2 = tk.Canvas( base\_frame\_tab\_2, width = 1250, height = 5000 )

base\_canvas\_tab\_2.pack( side = tk.LEFT, fill = tk.BOTH, expand = 1 )

my\_scrollbar = ttk.Scrollbar( base\_frame\_tab\_2, orient = tk.VERTICAL, command = base\_canvas\_tab\_2.yview )

my\_scrollbar.pack( side = tk.RIGHT, fill = tk.Y )

base\_canvas\_tab\_2.configure( yscrollcommand = my\_scrollbar.set )

base\_canvas\_tab\_2.bind( '<Configure>', lambda e : base\_canvas\_tab\_2.configure( scrollregion = base\_canvas\_tab\_2.bbox( "all" ) ) )

tab\_frame\_2 = tk.Frame( base\_canvas\_tab\_2, width = 1280, height = 5000, bg = "#65A8E8" )

base\_canvas\_tab\_2.create\_window( (0, 0), window = tab\_frame\_2, anchor = "nw" )

# \_\_\_\_Stuff in the tab\_\_\_#

bg\_label\_2 = tk.Label( tab\_frame\_2, text = 'Ratings Vs Videos Downloaded', bg = '#65A8E8', font = ("Calibre", 30) )

bg\_label\_2.place( relx = 0.3, rely = 0.01 )

ratings\_graph\_img = ImageTk.PhotoImage( Image.open( 'Assets/Graphs/ratings\_bar\_graph.png' ) )

ratings\_graph\_img\_lbl = tk.Label( tab\_frame\_2, image = ratings\_graph\_img )

ratings\_graph\_img\_lbl.place( relx = 0.25, rely = 0.03 )

ratings\_graph\_img\_2 = ImageTk.PhotoImage( Image.open( 'Assets/Graphs/ratings\_hist\_graph.png' ) )

ratings\_graph\_img\_2\_lbl = tk.Label( tab\_frame\_2, image = ratings\_graph\_img\_2 )

ratings\_graph\_img\_2\_lbl.place( relx = 0.25, rely = 0.13 )

# \_\_\_#

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Tab 3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

base\_frame\_tab\_3 = tk.Frame( my\_notebook, width = 1280, height = 5000, bg = "#65A8E8" )

# Adding scrollbar to that tab

base\_canvas\_tab\_3 = tk.Canvas( base\_frame\_tab\_3, width = 1250, height = 5000 )

base\_canvas\_tab\_3.pack( side = tk.LEFT, fill = tk.BOTH, expand = 1 )

my\_scrollbar = ttk.Scrollbar( base\_frame\_tab\_3, orient = tk.VERTICAL, command = base\_canvas\_tab\_3.yview )

my\_scrollbar.pack( side = tk.RIGHT, fill = tk.Y )

base\_canvas\_tab\_3.configure( yscrollcommand = my\_scrollbar.set )

base\_canvas\_tab\_3.bind( '<Configure>', lambda e : base\_canvas\_tab\_3.configure( scrollregion = base\_canvas\_tab\_3.bbox( "all" ) ) )

tab\_frame\_3 = tk.Frame( base\_canvas\_tab\_3, width = 1280, height = 5000, bg = "#65A8E8" )

base\_canvas\_tab\_3.create\_window( (0, 0), window = tab\_frame\_3, anchor = "nw" )

# \_\_\_\_Stuff in the tab\_\_\_#

bg\_label\_3 = tk.Label( tab\_frame\_3, text = 'Likes Vs Videos Downloaded', bg = '#65A8E8', font = ("Calibre", 30) )

bg\_label\_3.place( relx = 0.3, rely = 0.01 )

likes\_graph\_img = ImageTk.PhotoImage( Image.open( 'Assets/Graphs/likes\_hist\_graph.png' ) )

likes\_graph\_img\_lbl = tk.Label( tab\_frame\_3, image = likes\_graph\_img )

likes\_graph\_img\_lbl.place( relx = 0.25, rely = 0.03 )

likes\_graph\_img\_2 = ImageTk.PhotoImage( Image.open( 'Assets/Graphs/likes\_line\_graph.png' ) )

likes\_graph\_img\_2\_lbl = tk.Label( tab\_frame\_3, image = likes\_graph\_img\_2 )

likes\_graph\_img\_2\_lbl.place( relx = 0.25, rely = 0.13 )

# \_\_\_#

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Tab 4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

base\_frame\_tab\_4 = tk.Frame( my\_notebook, width = 1280, height = 5000, bg = "#65A8E8" )

# Adding scrollbar to that tab

base\_canvas\_tab\_4 = tk.Canvas( base\_frame\_tab\_4, width = 1250, height = 5000 )

base\_canvas\_tab\_4.pack( side = tk.LEFT, fill = tk.BOTH, expand = 1 )

my\_scrollbar = ttk.Scrollbar( base\_frame\_tab\_4, orient = tk.VERTICAL, command = base\_canvas\_tab\_4.yview )

my\_scrollbar.pack( side = tk.RIGHT, fill = tk.Y )

base\_canvas\_tab\_4.configure( yscrollcommand = my\_scrollbar.set )

base\_canvas\_tab\_4.bind( '<Configure>', lambda e : base\_canvas\_tab\_4.configure( scrollregion = base\_canvas\_tab\_4.bbox( "all" ) ) )

tab\_frame\_4 = tk.Frame( base\_canvas\_tab\_4, width = 1280, height = 5000, bg = "#65A8E8" )

base\_canvas\_tab\_4.create\_window( (0, 0), window = tab\_frame\_4, anchor = "nw" )

# \_\_\_\_Stuff in the tab\_\_\_#

bg\_label\_4 = tk.Label( tab\_frame\_4, text = 'Dislikes Vs Videos Downloaded', bg = '#65A8E8', font = ("Calibre", 30) )

bg\_label\_4.place( relx = 0.3, rely = 0.01 )

dislikes\_graph\_img = ImageTk.PhotoImage( Image.open( 'Assets/Graphs/dislikes\_bar\_graph.png' ) )

dislikes\_graph\_img\_lbl = tk.Label( tab\_frame\_4, image = dislikes\_graph\_img )

dislikes\_graph\_img\_lbl.place( relx = 0.25, rely = 0.03 )

dislikes\_graph\_img\_2 = ImageTk.PhotoImage( Image.open( 'Assets/Graphs/dislikes\_hist\_graph.png' ) )

dislikes\_graph\_img\_2\_lbl = tk.Label( tab\_frame\_4, image = dislikes\_graph\_img\_2 )

dislikes\_graph\_img\_2\_lbl.place( relx = 0.25, rely = 0.13 )

# \_\_\_#

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Tab 5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_#

base\_frame\_tab\_5 = tk.Frame( my\_notebook, width = 1280, height = 5000, bg = "#65A8E8" )

# Adding scrollbar to that tab

base\_canvas\_tab\_5 = tk.Canvas( base\_frame\_tab\_5, width = 1250, height = 5000 )

base\_canvas\_tab\_5.pack( side = tk.LEFT, fill = tk.BOTH, expand = 1 )

my\_scrollbar = ttk.Scrollbar( base\_frame\_tab\_5, orient = tk.VERTICAL, command = base\_canvas\_tab\_5.yview )

my\_scrollbar.pack( side = tk.RIGHT, fill = tk.Y )

base\_canvas\_tab\_5.configure( yscrollcommand = my\_scrollbar.set )

base\_canvas\_tab\_5.bind( '<Configure>', lambda e : base\_canvas\_tab\_5.configure( scrollregion = base\_canvas\_tab\_5.bbox( "all" ) ) )

tab\_frame\_5 = tk.Frame( base\_canvas\_tab\_5, width = 1280, height = 5000, bg = "#65A8E8" )

base\_canvas\_tab\_5.create\_window( (0, 0), window = tab\_frame\_5, anchor = "nw" )

# \_\_\_\_Stuff in the tab\_\_\_#

bg\_label\_5 = tk.Label( tab\_frame\_5, text = 'Categories Vs Videos Downloaded', bg = '#65A8E8', font = ("Calibre", 30) )

bg\_label\_5.place( relx = 0.3, rely = 0.01 )

categories\_graph\_img = ImageTk.PhotoImage( Image.open( 'Assets/Graphs/categories\_bar\_graph.png' ) )

categories\_graph\_img\_lbl = tk.Label( tab\_frame\_5, image = categories\_graph\_img )

categories\_graph\_img\_lbl.place( relx = 0.25, rely = 0.03 )

categories\_graph\_img\_2 = ImageTk.PhotoImage( Image.open( 'Assets/Graphs/categories\_pie\_chart.png' ) )

categories\_graph\_img\_2\_lbl = tk.Label( tab\_frame\_5, image = categories\_graph\_img\_2 )

categories\_graph\_img\_2\_lbl.place( relx = 0.25, rely = 0.13 )

# \_\_\_#

# \_\_\_\_\_\_\_\_\_\_\_\_End of declaring Tabs\_\_\_\_\_\_\_\_\_\_\_\_\_#

# \_\_\_\_\_\_\_\_\_\_\_Adding them to the notebook, and activating them\_\_\_\_\_\_\_\_\_#

base\_frame\_tab\_1.pack( fill = "both", expand = 1 )

base\_frame\_tab\_2.pack( fill = "both", expand = 1 )

base\_frame\_tab\_3.pack( fill = "both", expand = 1 )

base\_frame\_tab\_4.pack( fill = "both", expand = 1 )

base\_frame\_tab\_5.pack( fill = "both", expand = 1 )

my\_notebook.add( base\_frame\_tab\_1, text = "Views" )

my\_notebook.add( base\_frame\_tab\_2, text = "Ratings" )

my\_notebook.add( base\_frame\_tab\_3, text = "Likes" )

my\_notebook.add( base\_frame\_tab\_4, text = "Dislikes" )

my\_notebook.add( base\_frame\_tab\_5, text = "Categories" )

root.mainloop()

class loading(tk.Tk):

def \_\_init\_\_(self, \*args, \*\*kwargs):

tk.Tk.\_\_init\_\_(self, \*args, \*\*kwargs)

self.geometry('445x245')

BG\_IMG = "Assets/Background Images/metadata\_img.png"

self.imgg = tk.PhotoImage( file = BG\_IMG, master = self )

self.label = tk.Label( self, image = self.imgg )

self.label.place(relx = -0.01, rely = -0.01)

self.progress = ttk.Progressbar(self, orient="horizontal",

length=450 , mode="determinate", style="TProgressbar")

self.progress.place(rely = 0.65, relx = -0.01, relheight = 0.08)

self.bytes = 0

self.maxbytes = 0

self.start()

def start(self):

self.progress["value"] = 0

self.maxbytes = int(1.5\*len(playlist\_URLS))

self.progress["maximum"] = int(1.5\*len(playlist\_URLS))

self.read\_bytes()

def read\_bytes(self):

'''simulate reading 500 bytes; update progress bar'''

self.bytes += 0.1

self.progress["value"] = self.bytes

if self.bytes < self.maxbytes:

# read more bytes after 100 ms

self.after(100, self.read\_bytes)

if done:

self.destroy()

else:

while True:

if done:

self.destroy()

break

def generate\_vids() :

global done, video\_titles, video\_titles\_with\_urls

video\_titles\_with\_urls = [[] for i in range(len(playlist\_URLS))]

print(video\_titles\_with\_urls)

ydl = youtube\_dl.YoutubeDL({'outtmpl': '%(id)s%(ext)s', 'quiet':True,})

with ydl:

result = ydl.extract\_info(URL,download=False)

print('done')

if 'entries' in result:

video = result['entries']

j = 0

for i, \_ in enumerate(video):

video\_titles.append(result['entries'][i]['title'])

video\_titles\_with\_urls[j].append(video\_titles[j])

video\_titles\_with\_urls[j].append(playlist\_URLS[j])

print(video\_titles\_with\_urls)

j = j+1

fio.write.add\_playlist\_to\_csv(video, len(playlist\_URLS))

done = True

print(like\_counts, dislike\_counts)

# Function to run all the things. Function to return to. Function that calls. Function that manages.

def main() :

global again, playlist\_URLS

while again :

# Graphs.plot\_ratings\_vs\_videos()

# Graphs.plot\_views\_vs\_videos()

again = False

window.intro\_win() # gets the URL

if TYPE == 'SINGLE' :

# youtube object used to getting info that is common to both single and playlist downlaods

yt = pt.YouTube( URL )

fio.write.add\_to\_data\_csv( yt, URL )

window.sel\_download\_win\_single( URL, yt )

elif TYPE == 'PLAYLIST' :

playlist = pt.Playlist( URL )

playlist.\_video\_regex = re.compile( r"\"url\":\"(/watch\?v=[\w-]\*)" )

playlist\_URLS = playlist.video\_urls

print(playlist\_URLS[0])

T1 = threading.Thread( target = generate\_vids )

T1.start()

app = loading()

T2 = threading.Thread( target = app.mainloop() )

T2.start()

window.sel\_downlaod\_win\_playlist( playlist )

elif TYPE == 'STATISTICS' :

window.statistics()

if not again :

print( 'Thanks for using Kappa video downloader' )

main()

"""

This file is used to define functions that are going to take in a youtube object, and then write the data onto files.

"""

# import pytube as pt

import youtube\_dl, pandas as pd, numpy as np

ydl\_opts = {}

def conv\_len(length):

minutes\_or\_hours = length // 60 # 563

if minutes\_or\_hours < 60:

minutes = minutes\_or\_hours

seconds = length - (60 \* minutes)

if minutes < 10:

vid\_len = "00:0" + str(minutes) + ":" + str(seconds)

else:

vid\_len = "00:0" + str(minutes) + ":" + str(seconds)

return vid\_len

else:

hours = minutes\_or\_hours // 60

minutes = minutes\_or\_hours - (60 \* hours)

our\_seconds = hours \* 3600 + minutes \* 60

seconds = length - our\_seconds

if minutes < 10 and seconds < 10:

vid\_len = str(hours) + ":0" + str(minutes) + ":0" + str(seconds)

elif minutes < 10 and seconds > 10:

vid\_len = str(hours) + ":0" + str(minutes) + ":" + str(seconds)

else:

vid\_len = str(hours) + ":" + str(minutes) + ":" + str(seconds)

return vid\_len

class write:

@staticmethod

def add\_to\_data\_csv(video, url):

with youtube\_dl.YoutubeDL(ydl\_opts) as ydl:

meta = ydl.extract\_info(url, download=False)

video\_likes = meta["like\_count"]

video\_dislikes = meta["dislike\_count"]

video\_category = meta["categories"][0]

video\_date = meta["upload\_date"]

new\_entry = True

initial = read.get\_df\_from\_csv()

titles = initial["video\_title"]

for i in range(len(titles)):

if video.title == titles[i]:

new\_entry = False

break

if new\_entry:

data = {

"video\_title": pd.Series([video.title], index=[0]),

"video\_views": pd.Series([video.views], index=[0]),

"video\_dislikes": pd.Series([video\_dislikes], index=[0]),

"video\_likes": pd.Series([video\_likes], index=[0]),

"video\_rating": pd.Series([video.rating], index=[0]),

"video\_length": pd.Series([conv\_len(video.length)], index=[0]),

"video\_category": pd.Series([video\_category], index=[0]),

"video\_author": pd.Series([video.author], index=[0]),

"video\_publish\_date": pd.Series([video\_date], index=[0]),

}

df = pd.DataFrame(data)

df = pd.concat([initial, df], ignore\_index=True)

df.to\_csv("Data/video\_data.csv", index=False)

@staticmethod

def add\_playlist\_to\_csv(video, number):

video\_titles = []

video\_ratings = []

video\_categories = []

video\_publish\_dates = []

video\_authors = []

video\_likes = []

video\_dislikes = []

video\_views = []

video\_lengths = []

initial = read.get\_df\_from\_csv()

for i, item in enumerate(video):

new\_entry = True

single\_vid\_title = video[i]["title"]

titles = initial["video\_title"]

# Check if the video already exists.

for j in range(len(titles)):

if single\_vid\_title == titles[j]:

return

video\_titles.append(video[i]["title"])

video\_ratings.append(video[i]["average\_rating"])

video\_categories.append(video[i]["categories"])

video\_publish\_dates.append(video[i]["upload\_date"])

video\_authors.append(video[i]["uploader"])

video\_likes.append(video[i]["like\_count"])

video\_dislikes.append(video[i]["dislike\_count"])

video\_views.append(video[i]["view\_count"])

video\_lengths.append(conv\_len(int(video[i]["duration"])))

data = {

'video\_title' : pd.Series( video\_titles , index = np.arange(number) ),

'video\_views' : pd.Series( video\_views , index = np.arange(number) ),

'video\_dislikes' : pd.Series( video\_dislikes , index = np.arange(number) ),

'video\_likes' : pd.Series( video\_likes , index = np.arange(number) ),

'video\_rating' : pd.Series( video\_ratings , index = np.arange(number) ),

'video\_length' : pd.Series( video\_lengths , index = np.arange(number) ),

'video\_category' : pd.Series( video\_categories , index = np.arange(number) ),

'video\_author' : pd.Series( video\_authors , index = np.arange(number) ),

'video\_publish\_date' : pd.Series( video\_publish\_dates , index = np.arange(number) ),

}

df = pd.DataFrame(data)

df = pd.concat([initial, df], ignore\_index=True)

df.to\_csv("Data/video\_data.csv", index=False)

class read:

@staticmethod

def get\_titles():

df = read.get\_df\_from\_csv()

return df["video\_title"].tolist()

@staticmethod

def get\_views():

df = read.get\_df\_from\_csv()

return df["video\_views"].tolist()

@staticmethod

def get\_ratings():

df = read.get\_df\_from\_csv()

return df["video\_rating"].tolist()

@staticmethod

def get\_publish\_dates():

df = read.get\_df\_from\_csv()

return df["video\_publish\_date"].tolist()

@staticmethod

def get\_lengths():

df = read.get\_df\_from\_csv()

return df["video\_length"].tolist()

@staticmethod

def get\_authors():

df = read.get\_df\_from\_csv()

return df["video\_author"].tolist()

@staticmethod

def get\_categories():

df = read.get\_df\_from\_csv()

return df["video\_category"].tolist()

@staticmethod

def get\_likes():

df = read.get\_df\_from\_csv()

return df["video\_likes"].tolist()

@staticmethod

def get\_dislikes():

df = read.get\_df\_from\_csv()

return df["video\_dislikes"].tolist()

@staticmethod

def get\_df\_from\_csv():

return pd.read\_csv("Data/video\_data.csv")

import pytube as pt

tags = {

313: '2160p',

271: '1440p',

137: '1080p',

22: '720p',

18: '360p',

278: '144p',

140: 'MP4 128kb/s (only audio)',

249: 'webm 50Kb/s (only audio)',

251: 'webm 160kb/s (onlu audio)',

}

def get\_available\_qualities(url):

video = pt.YouTube(url)

qualities = []

for i in tags.keys():

if video.streams.get\_by\_itag(i):

qualities.append(tags.get(i))

return qualities

def get\_available\_qualities\_with\_obj(youtube\_obj):

qualities = []

for i in tags.keys():

if youtube\_obj.streams.get\_by\_itag(i):

qualities.append(tags.get(i))

return qualities

"""

This file has all the general methods that are used here and there in the window program

"""

import urllib.request

import os

# gets the id of the video from the url, this id is used to store the thumbnail of the video later on

def get\_vid\_id( url ) :

return url[ url.index( "=" ) + 1 : ]

# Gets the video thumbnail and saves it in the correct folder

def get\_video\_tnl( url, tnurl ) :

vid\_id = get\_vid\_id( url ) + '.png'

tnl = urllib.request.urlretrieve( tnurl, os.path.join( 'Assets/Thumbnails', vid\_id ) )

return tnl

# Convers the lengths of the videos from seconds to displayable and understandable formats

def conv\_len( length ) :

print( length )

minutes\_or\_hours = length // 60 # 563

if minutes\_or\_hours < 60 :

print( minutes\_or\_hours )

minutes = minutes\_or\_hours

seconds = length - (60 \* minutes)

if minutes < 10 :

vid\_len = '00:0' + str( minutes ) + ':' + str( seconds )

else :

vid\_len = '00:0' + str( minutes ) + ':' + str( seconds )

return vid\_len

else :

print( minutes\_or\_hours )

hours = minutes\_or\_hours // 60

print( 'hours', hours )

minutes = (minutes\_or\_hours - (60 \* hours))

our\_seconds = hours \* 3600 + minutes \* 60

seconds = length - our\_seconds

if minutes < 10 and seconds < 10 :

vid\_len = str( hours ) + ':0' + str( minutes ) + ':0' + str( seconds )

elif minutes < 10 and seconds > 10 :

vid\_len = str( hours ) + ':0' + str( minutes ) + ':' + str( seconds )

else :

vid\_len = str( hours ) + ':' + str( minutes ) + ':' + str( seconds )

return vid\_len

# here is an example graph

import numpy as np

import File\_IO as fio

import matplotlib.pyplot as plt

def plot\_views\_vs\_videos():

"""

Plots a graph by looking at the data stored in the files, and then saves the graph in Assets/Graphs as png.

None -> None

"""

views\_list = fio.read.get\_views()

plt.xlabel('Videos')

plt.ylabel('views')

plt.bar(np.arange(len(views\_list)), views\_list, color = 'green')

plt.savefig('Assets/Graphs/views\_bar\_graph.png')

def plot\_views\_vs\_videos\_line():

"""

Plots a graph by looking at the data stored in the files, and then saves the graph in Assets/Graphs as png.

None -> None

"""

views\_list = fio.read.get\_views()

for i in range(len(views\_list)):

if views\_list[i] > 3e6:

views\_list[i] = 3e6

plt.xlabel('Videos')

plt.ylabel('views')

plt.plot(np.arange(len(views\_list)), views\_list, color = 'green')

plt.savefig('Assets/Graphs/views\_line\_graph.png')

def plot\_views\_vs\_videos\_hist():

"""

Plots a graph by looking at the data stored in the files, and then saves the graph in Assets/Graphs as png.

None -> None

"""

views\_list = fio.read.get\_views()

bin\_val = 1000

bins = []

Max = 0

for i in views\_list:

if i > Max:

Max = i

for i in range(15):

bins.append(i\*bin\_val)

plt.hist(views\_list, bins, histtype='bar', rwidth=0.8, color = 'yellow')

plt.xlabel('Views')

plt.ylabel('Number of videos')

plt.title('number of videos with views')

plt.savefig('Assets/Graphs/views\_hist\_graph.png')

def plot\_likes\_vs\_videos\_hist():

"""

Plots a graph by looking at the data stored in the files, and then saves the graph in Assets/Graphs as png.

None -> None

"""

# this list is the raw list from the file, contains '\n' at the end that we don't need

video\_likes = fio.read.get\_likes()

for i in range(len(video\_likes)):

if video\_likes[i ] is None :

video\_likes[i] = 0

bin\_val = 1000

bins = []

Max = 0

for i in video\_likes:

if i > Max:

Max = i

for i in range(15):

bins.append(i\*bin\_val)

plt.hist(video\_likes, bins, histtype='bar', rwidth=0.8, color = 'yellow')

plt.xlabel('Views')

plt.ylabel('Number of videos')

plt.title('Number of videos vs likes')

plt.savefig('Assets/Graphs/likes\_hist\_graph.png')

def plot\_ratings\_vs\_videos():

"""

Plots a graph by looking at the data stored in the files, and then saves the graph in Assets/Graphs as png.

None -> None

"""

video\_ratings = fio.read.get\_ratings()

for i in range(len(video\_ratings)):

if video\_ratings[i ] is None :

video\_ratings[i] = 0.0

plt.xlabel('Videos')

plt.ylabel('Ratings')

# plots a simple graph that is saved as png in Assets/Graphs/. This file can then be accessed by other files.

plt.plot(np.arange(len(video\_ratings)), video\_ratings, color = 'red')

plt.savefig('Assets/Graphs/ratings\_bar\_graph.png')

def plot\_ratings\_vs\_videos\_hist():

"""

Plots a graph by looking at the data stored in the files, and then saves the graph in Assets/Graphs as png.

None -> None

"""

# this list is the raw list from the file, contains '\n' at the end that we don't need

video\_ratings = fio.read.get\_ratings()

for i in range(len(video\_ratings)):

if video\_ratings[i ] is None :

video\_ratings[i] = 0.0

bin\_val = 0.25

bins = []

for i in range(8):

bins.append(3+i\*bin\_val)

print(bins)

plt.hist(video\_ratings, bins, histtype='bar', rwidth=0.8, color = 'pink')

plt.xlabel('Views')

plt.ylabel('Number of videos')

plt.title('Number of videos vs ratings')

plt.savefig('Assets/Graphs/ratings\_hist\_graph.png')

def plot\_likes\_vs\_videos():

"""

Plots a graph by looking at the data stored in the files, and then saves the graph in Assets/Graphs as png.

None -> None

"""

# this list is the raw list from the file, contains '\n' at the end that we don't need

video\_likes = fio.read.get\_likes()

for i in range(len(video\_likes)):

if video\_likes[i ] is None :

video\_likes[i] = 0.0

for i in range(len(video\_likes)):

if video\_likes[i] > 5e5:

video\_likes[i] = 5e5

plt.xlabel('Videos')

plt.ylabel('likes')

# plots a simple graph that is saved as png in Assets/Graphs/. This file can then be accessed by other files.

plt.plot(np.arange(len(video\_likes)), video\_likes, color = 'green')

plt.savefig('Assets/Graphs/likes\_line\_graph.png')

def plot\_dislikes\_vs\_videos():

"""

Plots a graph by looking at the data stored in the files, and then saves the graph in Assets/Graphs as png.

None -> None

"""

video\_dislikes = fio.read.get\_dislikes()

for i in range(len(video\_dislikes)):

if video\_dislikes[i ] is None :

video\_dislikes[i] = 0.0

for i in range(len(video\_dislikes)):

if video\_dislikes[i] > 6e3:

video\_dislikes[i] = 6e3

plt.xlabel('Videos')

plt.ylabel('dislikes')

plt.bar(np.arange(len(video\_dislikes)), video\_dislikes, color = 'green')

plt.savefig('Assets/Graphs/dislikes\_bar\_graph.png')

def plot\_dislikes\_vs\_videos\_hist():

"""

Plots a graph by looking at the data stored in the files, and then saves the graph in Assets/Graphs as png.

None -> None

"""

# this list is the raw list from the file, contains '\n' at the end that we don't need

video\_dislikes = fio.read.get\_dislikes()

for i in range(len(video\_dislikes)):

if video\_dislikes[i ] is None :

video\_dislikes[i] = 0

bin\_val = 1000

bins = []

Max = 0

for i in video\_dislikes:

if i > Max:

Max = i

for i in range(15):

bins.append(i\*bin\_val)

plt.hist(video\_dislikes, bins, histtype='bar', rwidth=0.8, color = 'yellow')

plt.xlabel('Views')

plt.ylabel('Number of videos')

plt.title('Number of videos vs dislikes')

plt.savefig('Assets/Graphs/dislikes\_hist\_graph.png')

def plot\_categories\_vs\_videos():

"""

Plots a graph by looking at the data stored in the files, and then saves the graph in Assets/Graphs as png.

None -> None

"""

# this list is the raw list from the file, contains '\n' at the end that we do not need

video\_categories = fio.read.get\_categories()

cat\_list = ['Education', 'Science & Technology', 'Music', 'Autos & Vehicles', 'Entertainment', 'Howto & Style', 'People & Blogs']

cat\_list\_disp = ['Education', 'Science', 'Music', 'Vehicles', 'Entertain', 'How-to', 'People']

sorted\_cats = [0 for i in range(len(cat\_list))]

for i in range(len(cat\_list)):

for j in range(i, len(video\_categories)):

if cat\_list[i] == video\_categories[j]:

sorted\_cats[i] += 1

# plots a simple graph that is saved as png in Assets/Graphs/. This file can then be accessed by other files.

plt.bar(cat\_list\_disp, sorted\_cats, color = 'orange')

plt.savefig('Assets/Graphs/categories\_bar\_graph.png')

def plot\_categories\_vs\_videos\_pie():

"""

Plots a graph by looking at the data stored in the files, and then saves the graph in Assets/Graphs as png.

None -> None

"""

# this list is the raw list from the file, contains '\n' at the end that we do not need

video\_categories = fio.read.get\_categories()

cat\_list = ['Education', 'Science & Technology', 'Music', 'Autos & Vehicles', 'Entertainment', 'Howto & Style', 'People & Blogs']

cat\_list\_disp = ['Education', 'Science', 'Music', 'Vehicles', 'Entertain', 'How-to', 'People']

sorted\_cats = [0 for i in range(len(cat\_list))]

for i in range(len(cat\_list)):

for j in range(i, len(video\_categories)):

if cat\_list[i] == video\_categories[j]:

sorted\_cats[i] += 1

# plots a simple graph that is saved as png in Assets/Graphs/. This file can then be accessed by other files.

plt.pie(sorted\_cats, labels = cat\_list\_disp,explode = (0.1, 0, 0, 0, 0, 0, 0), shadow = True )

plt.savefig('Assets/Graphs/categories\_pie\_chart.png')

# plot\_views\_vs\_videos()

# plot\_ratings\_vs\_videos\_hist()

# plot\_views\_vs\_videos\_hist()

# plot\_views\_vs\_videos\_line()

# plot\_ratings\_vs\_videos()

# plot\_likes\_vs\_videos()

# plot\_likes\_vs\_videos\_hist()

# plot\_dislikes\_vs\_videos()

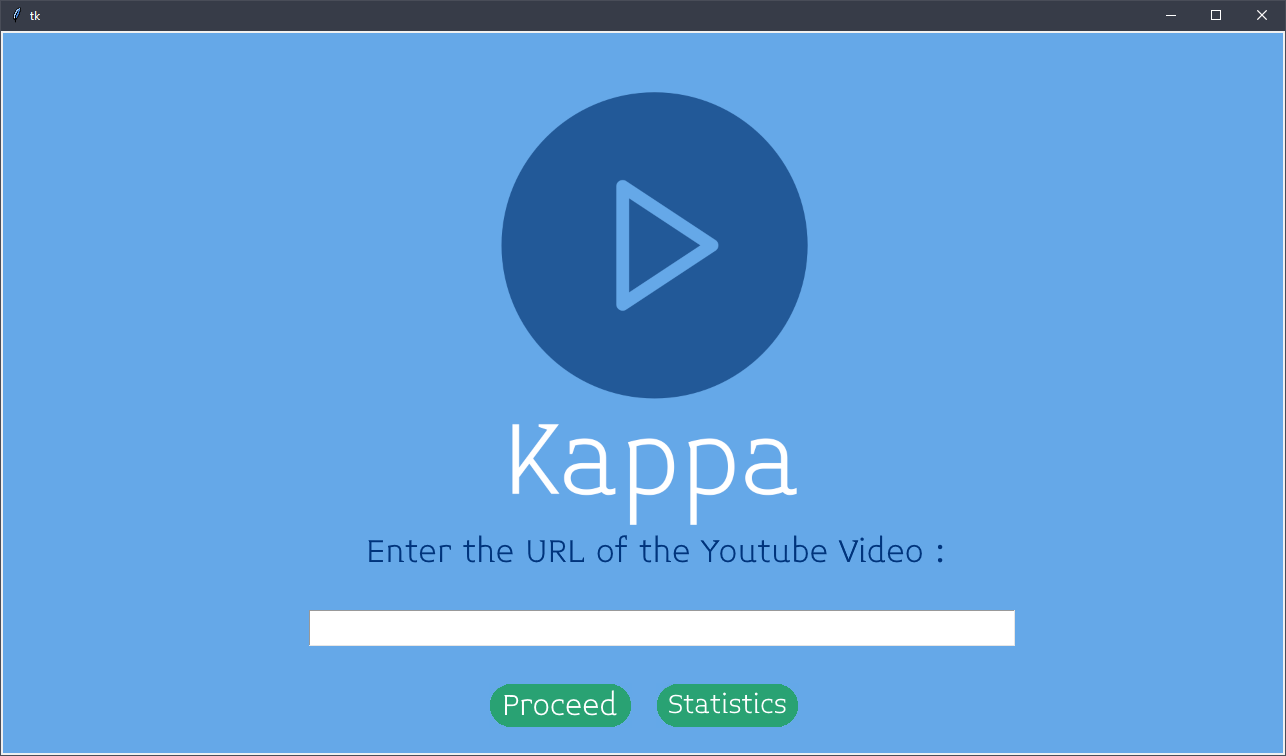
# plot\_dislikes\_vs\_videos\_hist()

# plot\_categories\_vs\_videos()

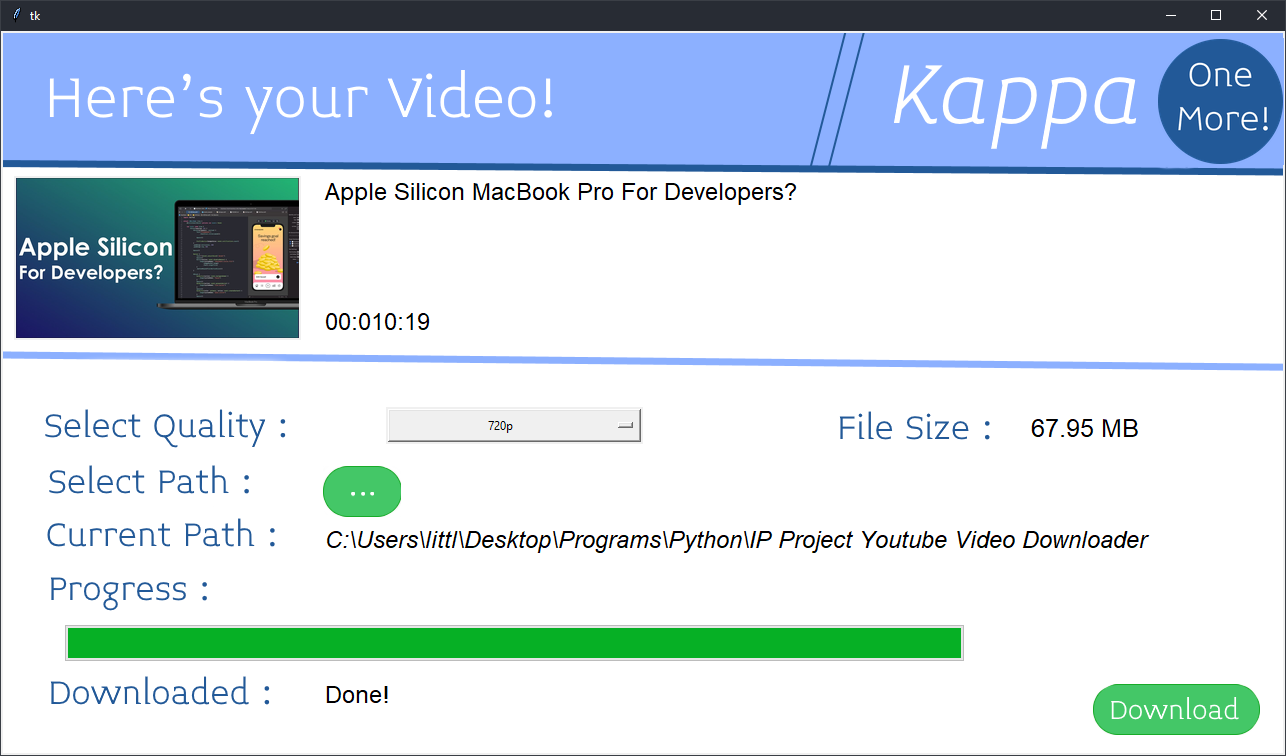
# plot\_categories\_vs\_videos\_pie()

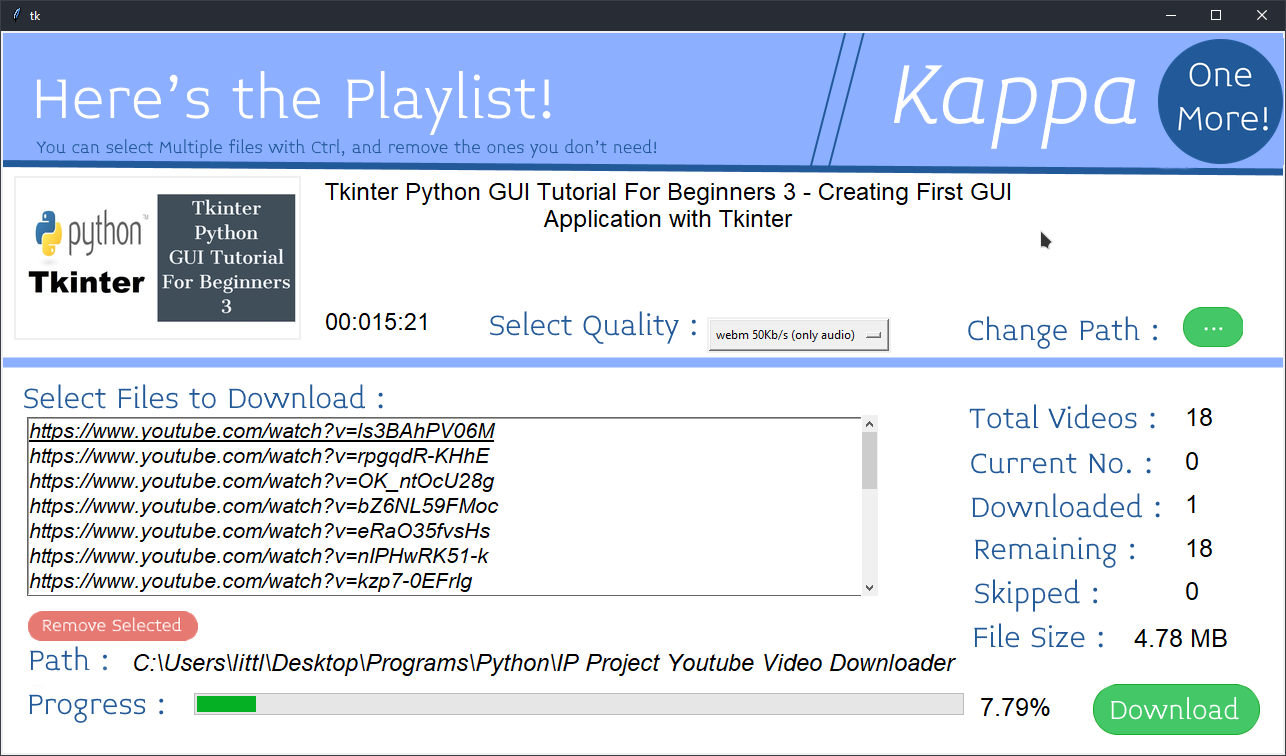
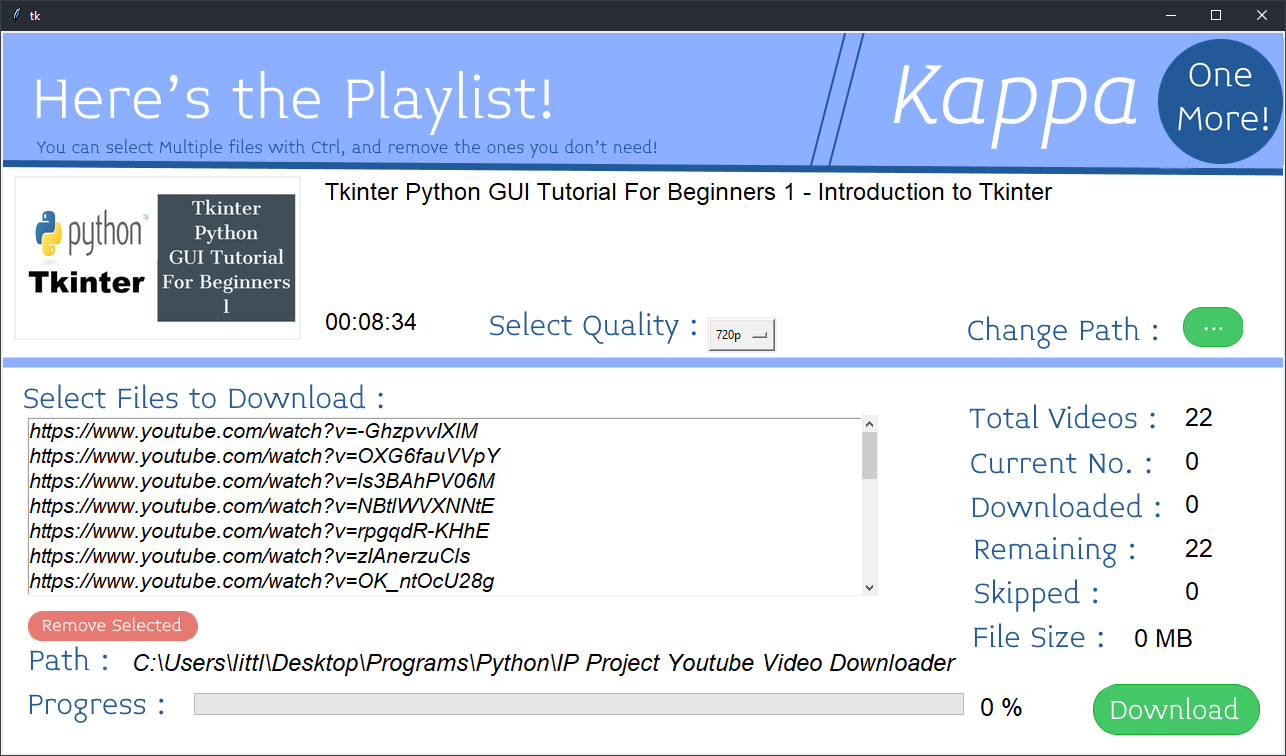
SCREENSHOTS OF THE PROJECT

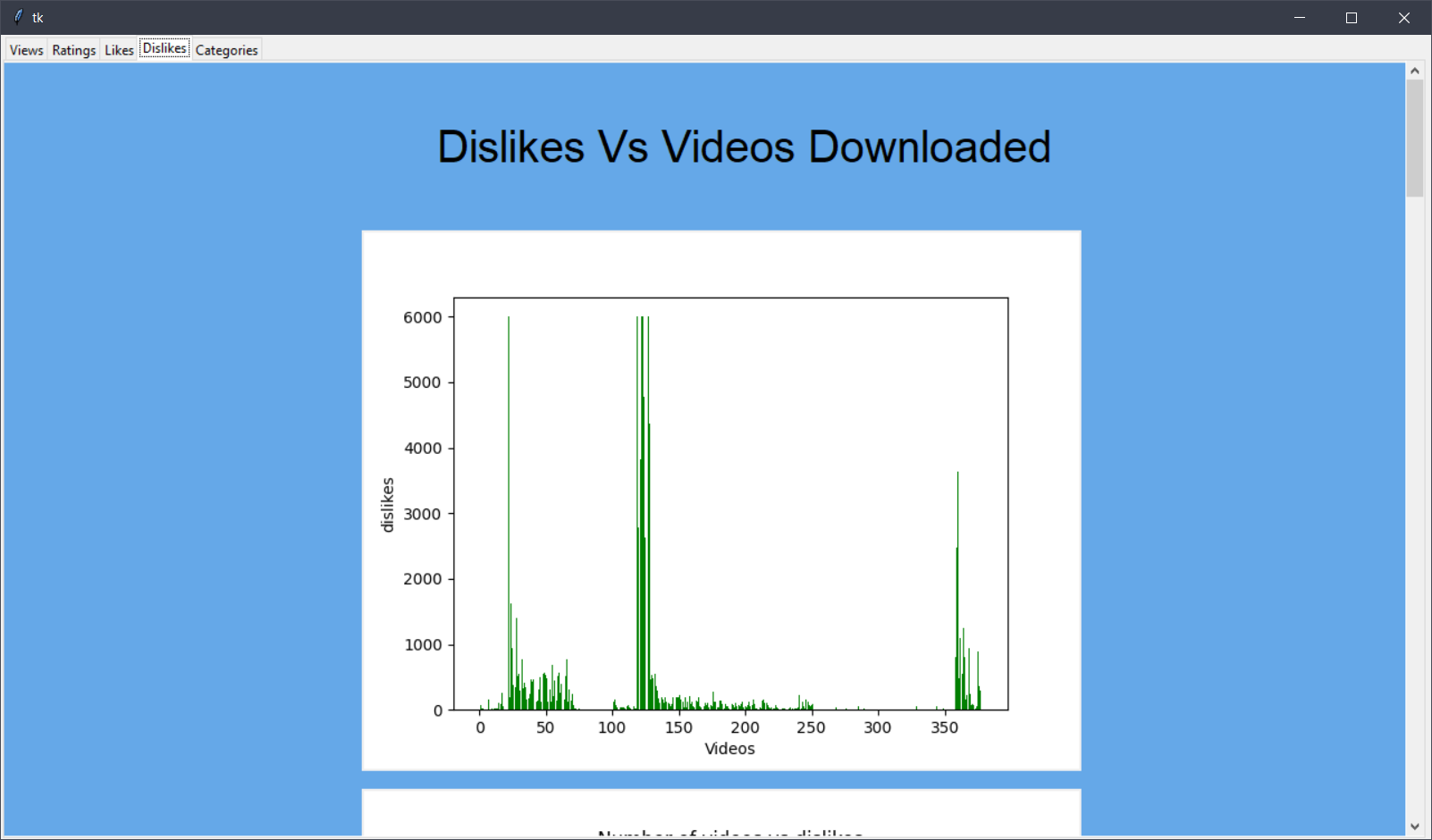
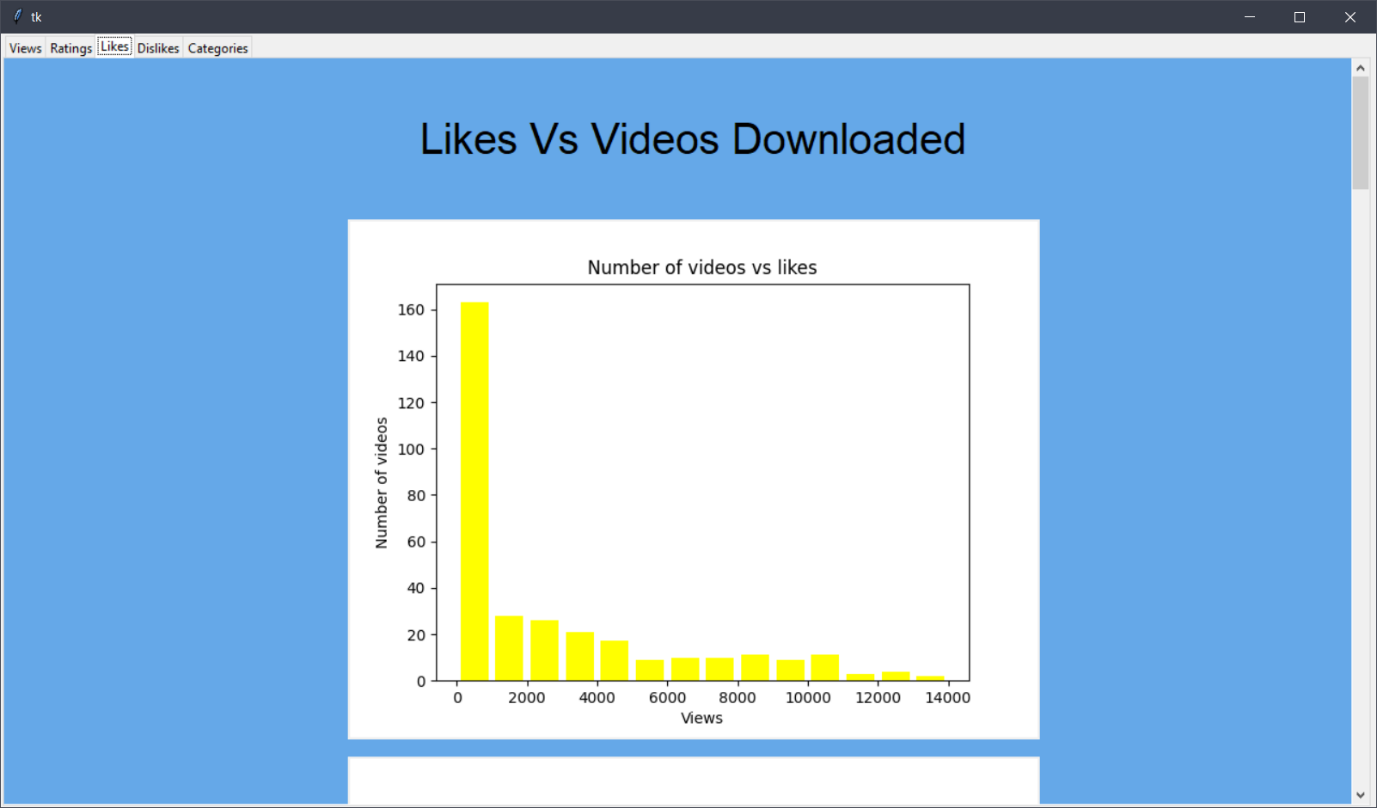
1. Main Menu Screen.

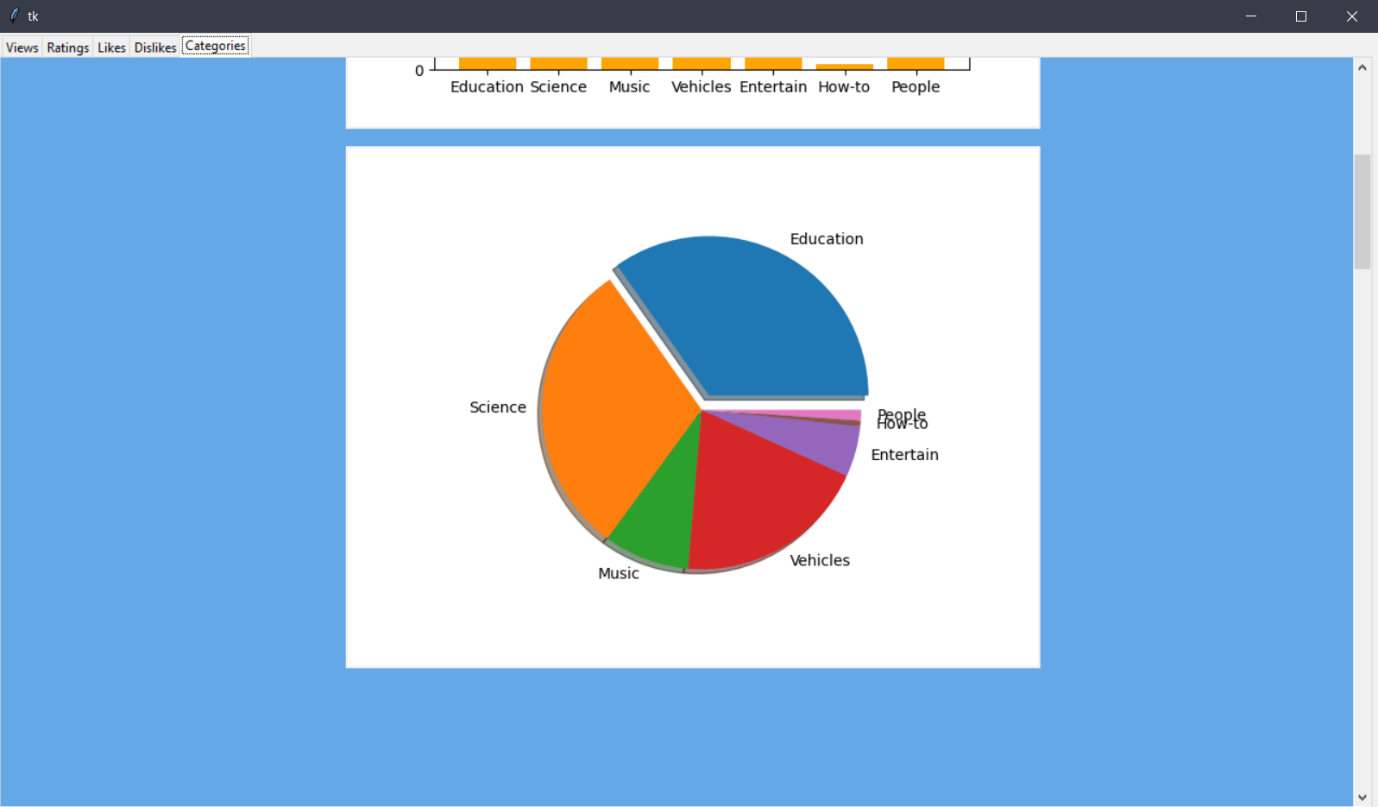
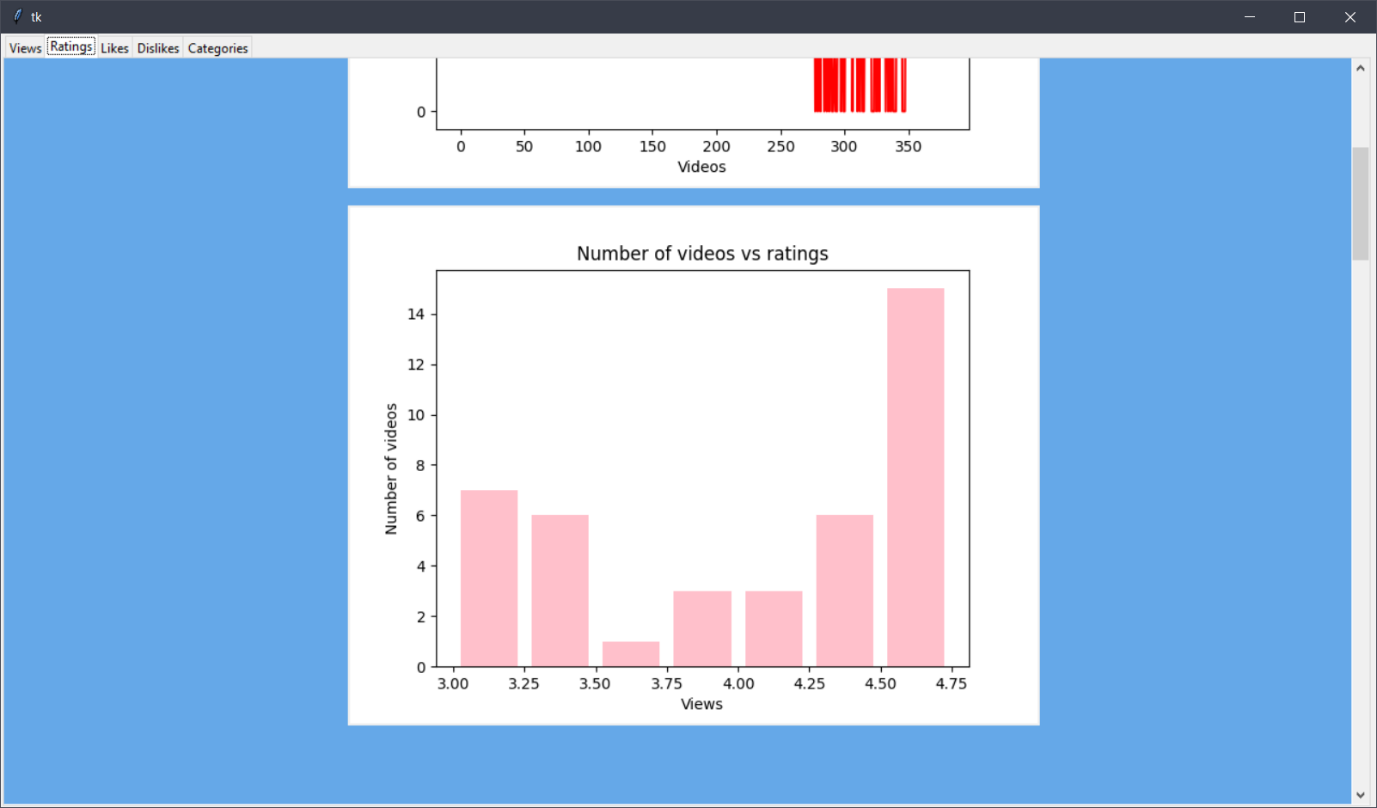


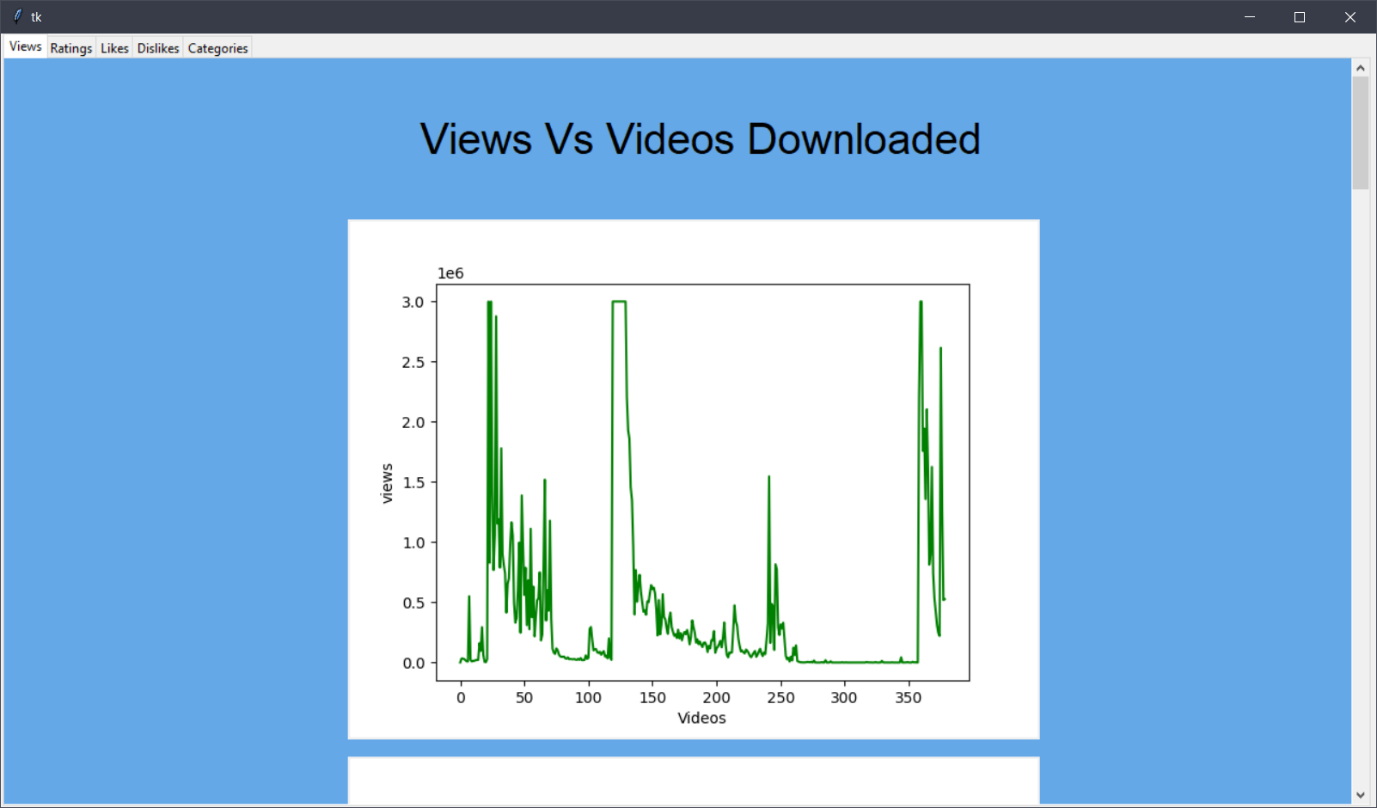
1. Downloading the Video

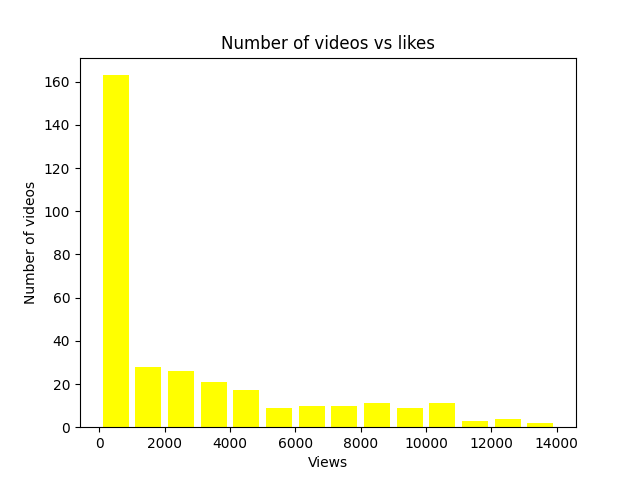
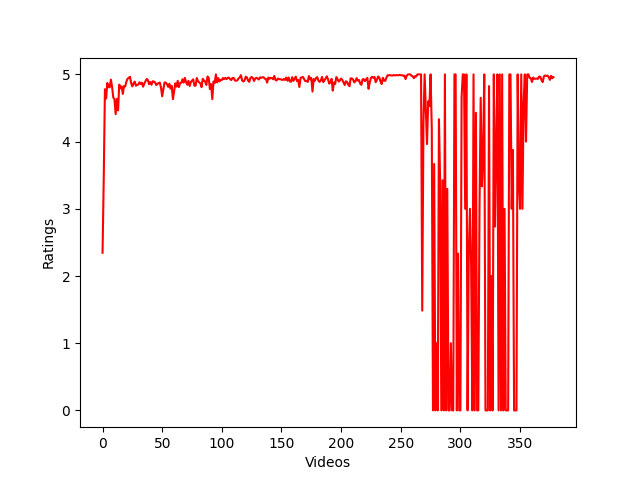


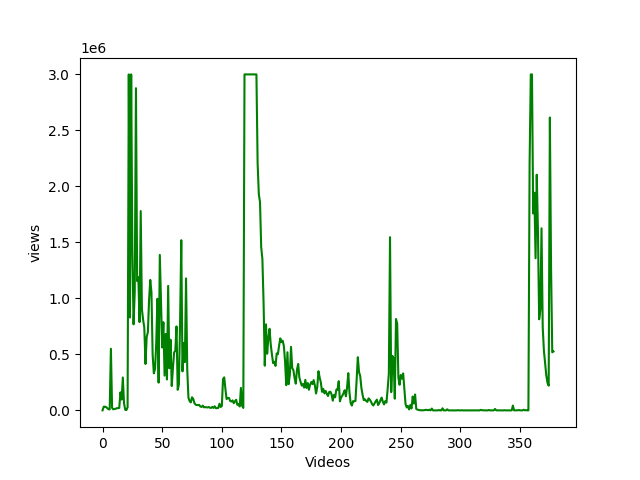
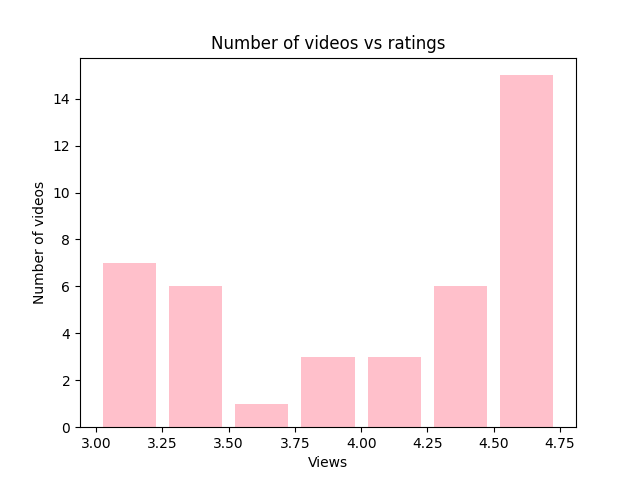












Bibliography

Thanks to Teachers, Parents, and Group members for their support and Guidance

Credits to https://youtube.com/Codemy for help on learning tkinter.

Other helpful websites:

1. <https://www.stackoverflow.com> for miscellaneous code and programming queries.

2. <https://www.github.com> for student communication

3. <https://www.namecheap.com> for logo development

Credits to the developers of modules like Pytube and youtube\_dl.

Other Important and useful software used for this project :

1. ShareX for taking screenshots
2. Paint.net for editing and making the background images and doing all of the graphics and paint work in the project
3. Visual studio code and Pycharm IDE community edition for the coding.
4. Github.com for communication between students.
5. Github Desktop for uploading the code and keeping it up to date.