COVER PAGE

CERTIFICATE

ACKNOWLEDGEMENT

INDEX

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 :

A. Download Quality :
   1. 1080p
   2. 720p
   3. 480p
   4. 2160p
   5. 120 Hz audio

B. Download Format :
   1. .mp4
   2. .webm
   3. .ogg
   4. .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

18. title
19. views
20. rating
21. author
22. description
23. Date of Publication
24. Likes to dislike ratio
25. 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')
```

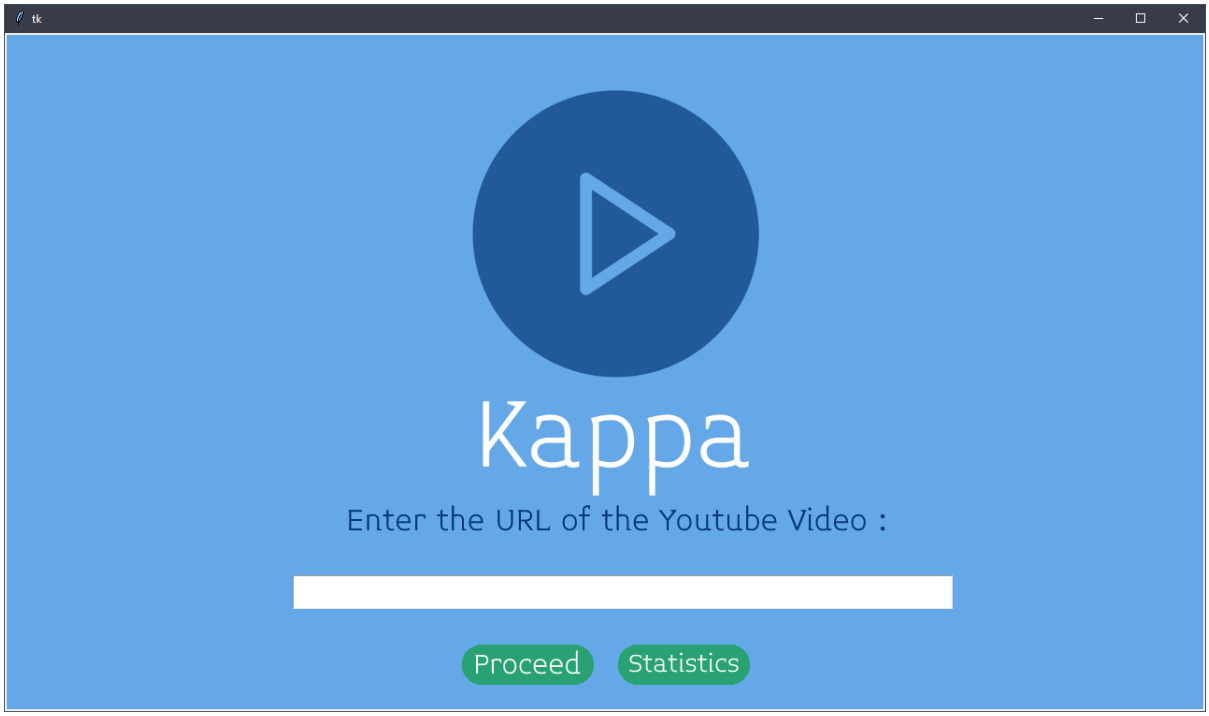
```
	# plots a simple graph that is saved as png in Assets/Graphs/.
This file can then be accessed by other files.
	plt.plot(np.arange(len(video_likes)), video_likes, color =
'green')
	plt.savefig('Assets/Graphs/likes_line_graph.png')

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))]
```

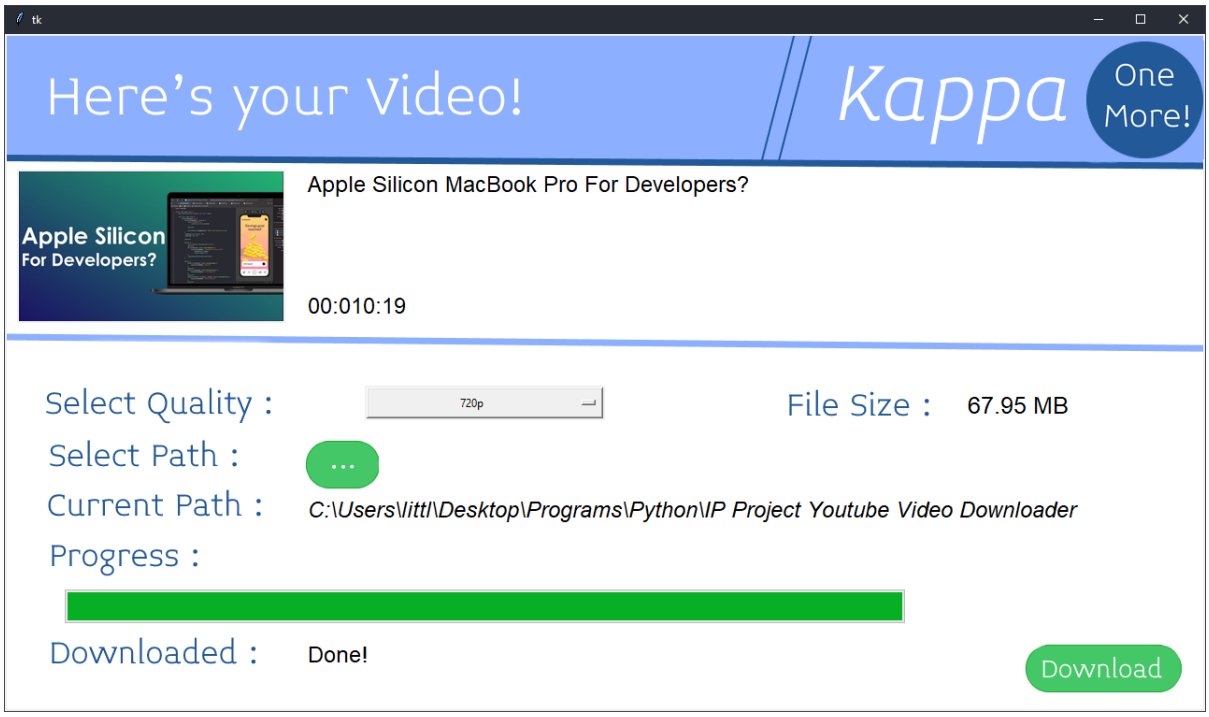
```
      for i in range(len(cat_list)):
            for j in range(i, len(video_categories)):
                  if cat_list[i] == video_categories[j]:
                        sorted_cats[i] += 1
      # plots a simple graph that is saved as png in Assets/Graphs/.
This file can then be accessed by other 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()
```

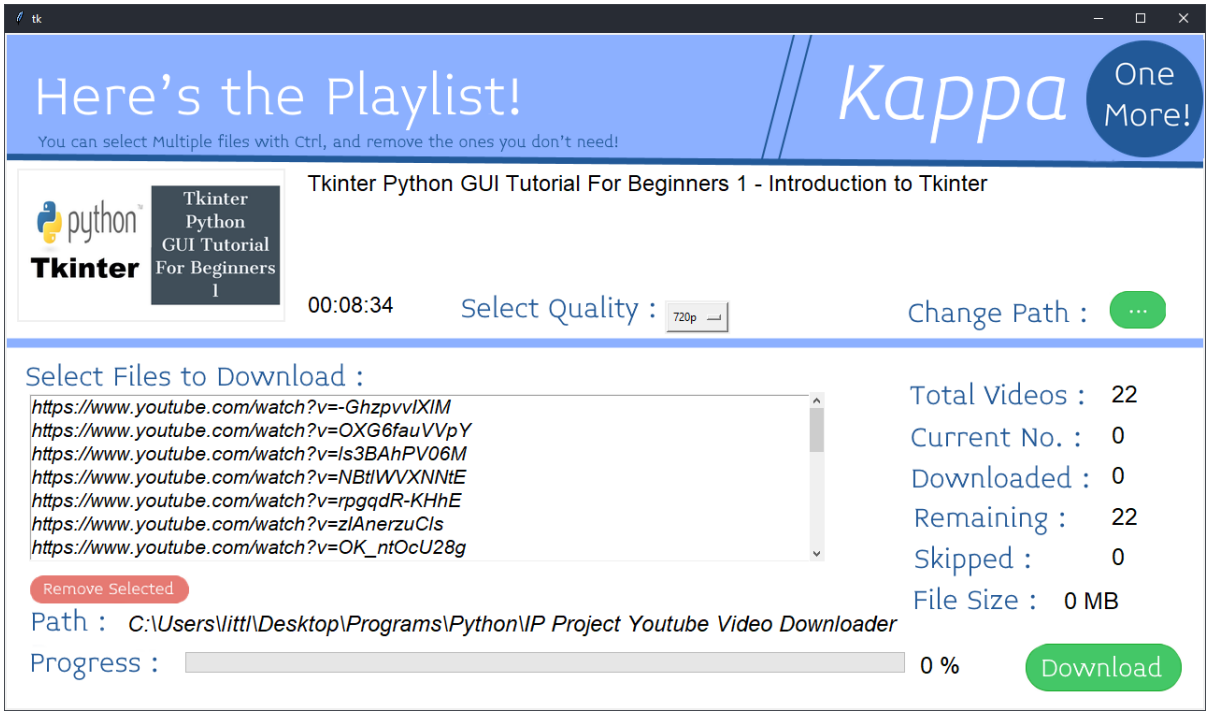
SCREENSHOTS OF THE PROJECT

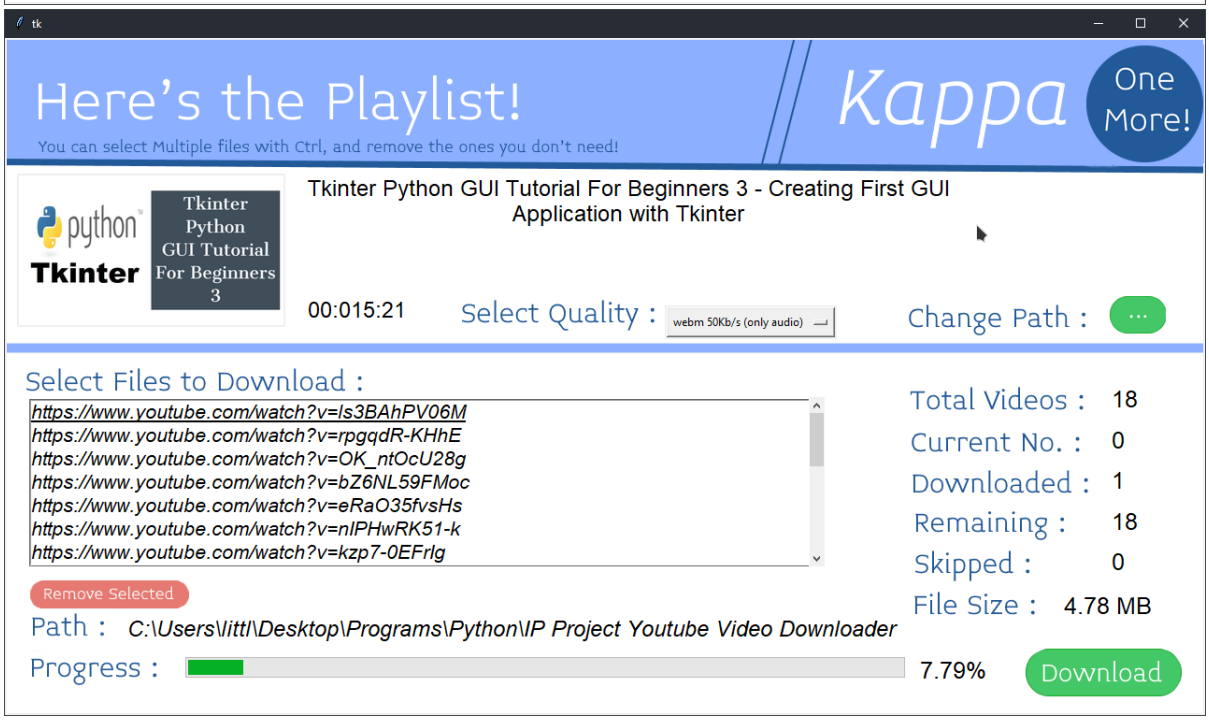
1. Main Menu Screen.

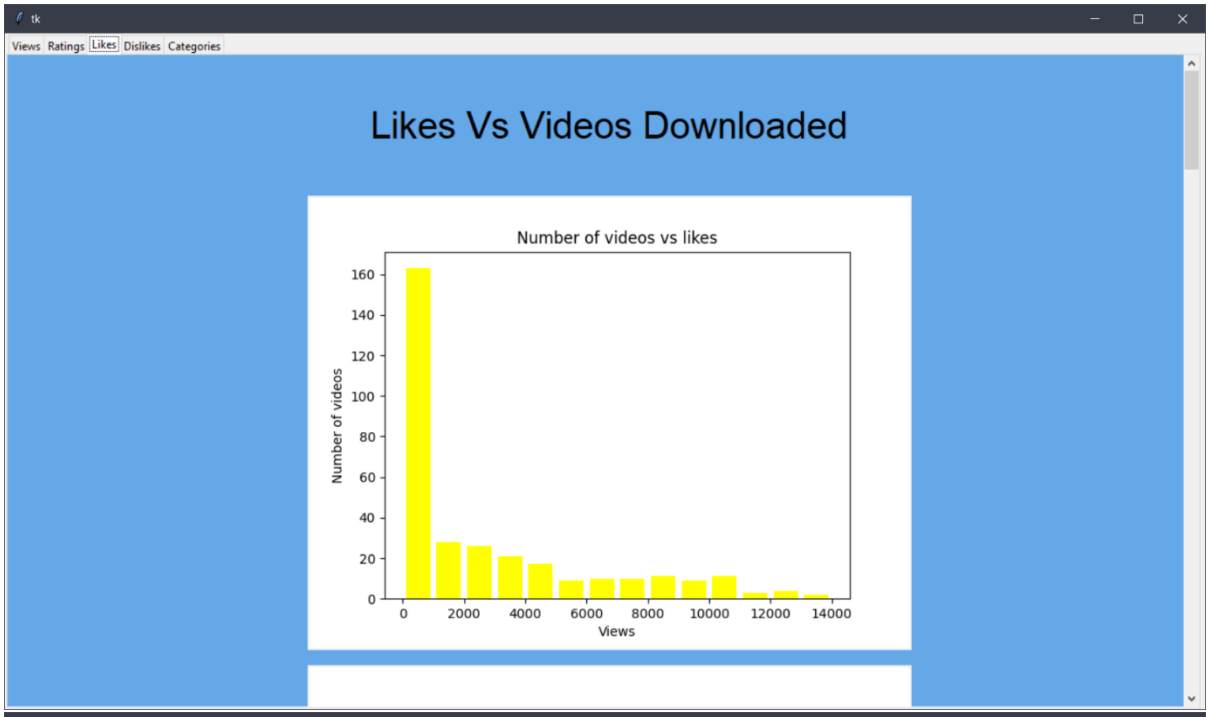


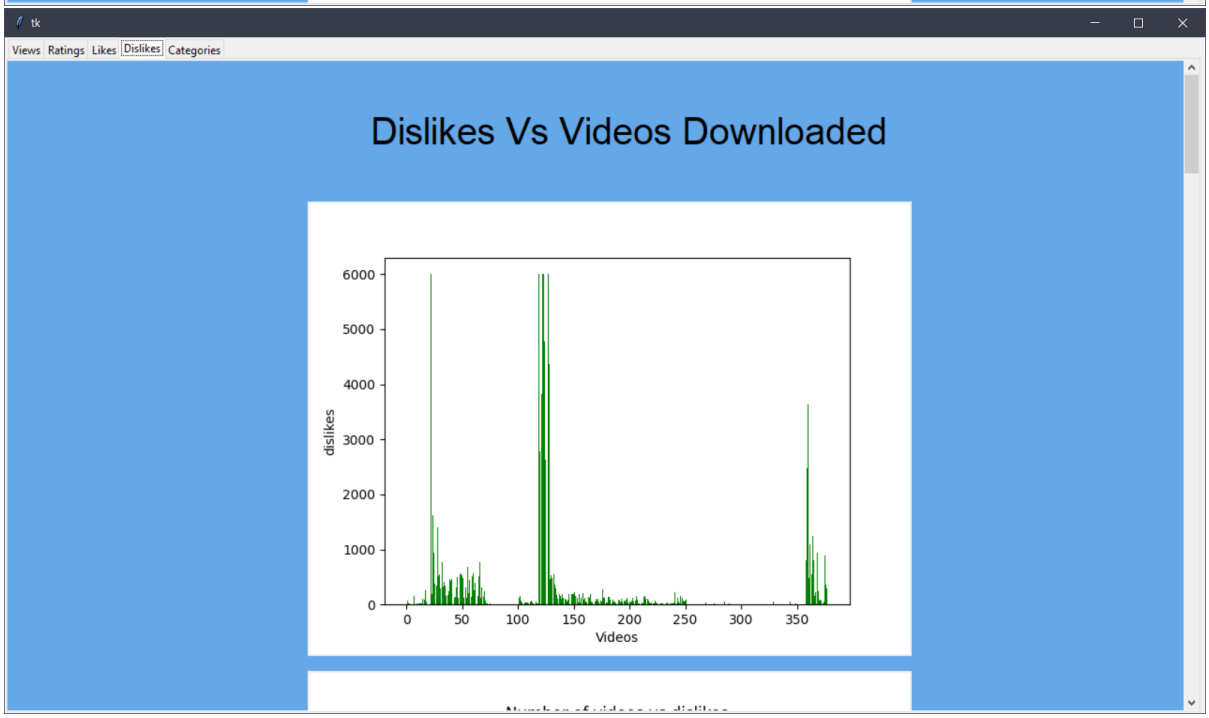
2. Downloading the Video

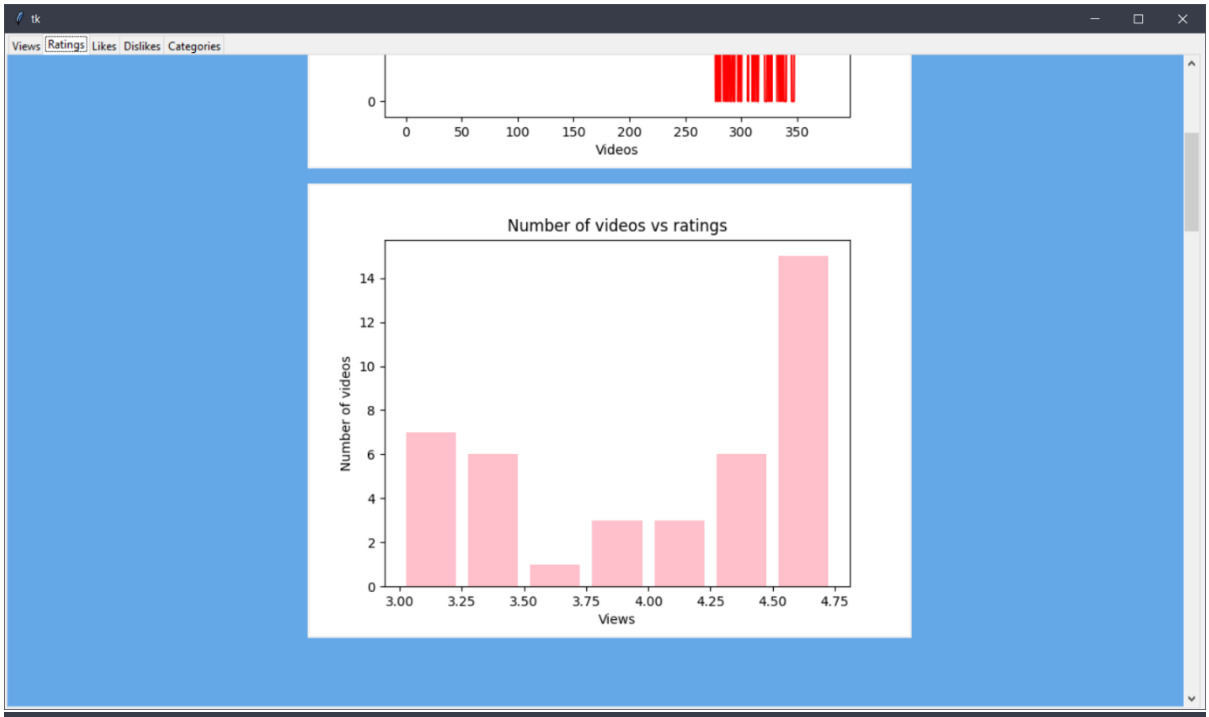


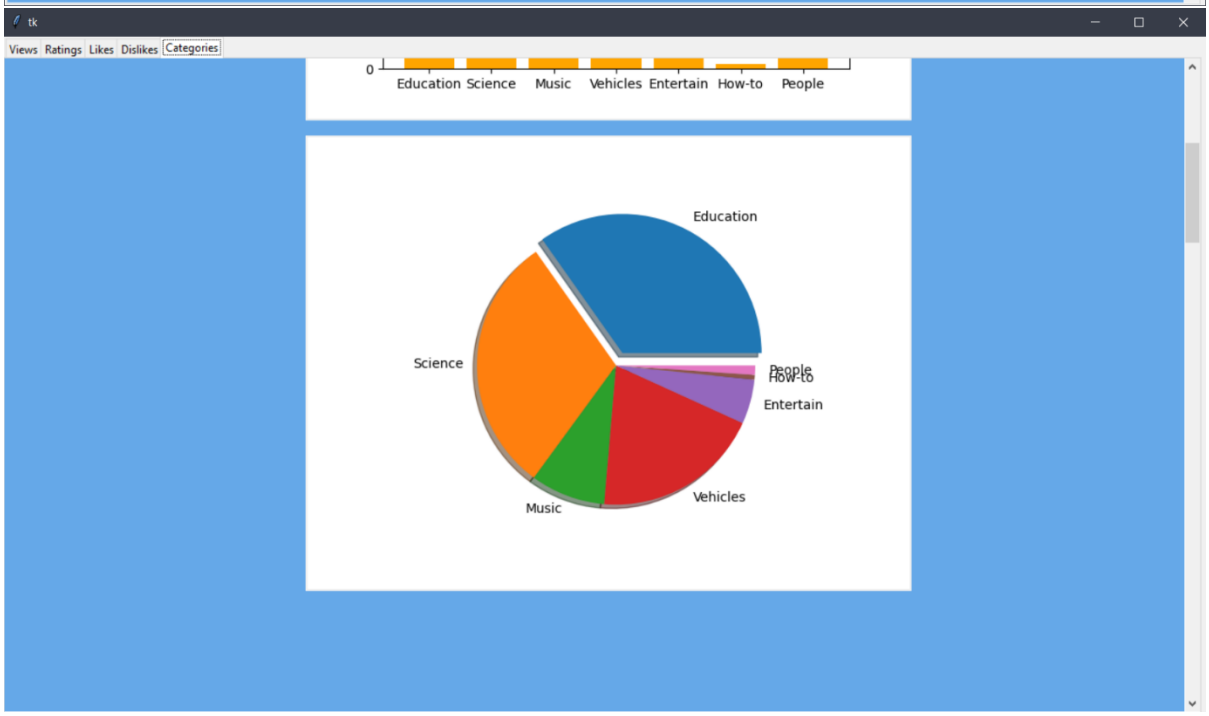

tk
Here's the Playlist!
You can select Multiple files with Ctrl, and remove the ones you don't need!
Kappa
One More!
python
Tkinter
Tkinter Python GUI Tutorial For Beginners 1
Tkinter Python GUI Tutorial For Beginners 1 - Introduction to Tkinter
00:08:34
Select Quality :
720p
Change Path :
Select Files to Download :
https://www.youtube.com/watch?v=-GhzpvvlXIM
https://www.youtube.com/watch?v=OXG6fauVVpY
https://www.youtube.com/watch?v=ls3BAhPV06M
https://www.youtube.com/watch?v=NBtlWVXNNtE
https://www.youtube.com/watch?v=rpgqdR-KHhE
https://www.youtube.com/watch?v=zlAnerzuCls
https://www.youtube.com/watch?v=OK_ntOcU28g
Total Videos : 22
Current No. : 0
Downloaded : 0
Remaining : 22
Skipped : 0
File Size : 0 MB
Remove Selected
Path : C:\Users\ittl\Desktop\Programs\Python\IP Project Youtube Video Downloader
Progress :
0 %
Download

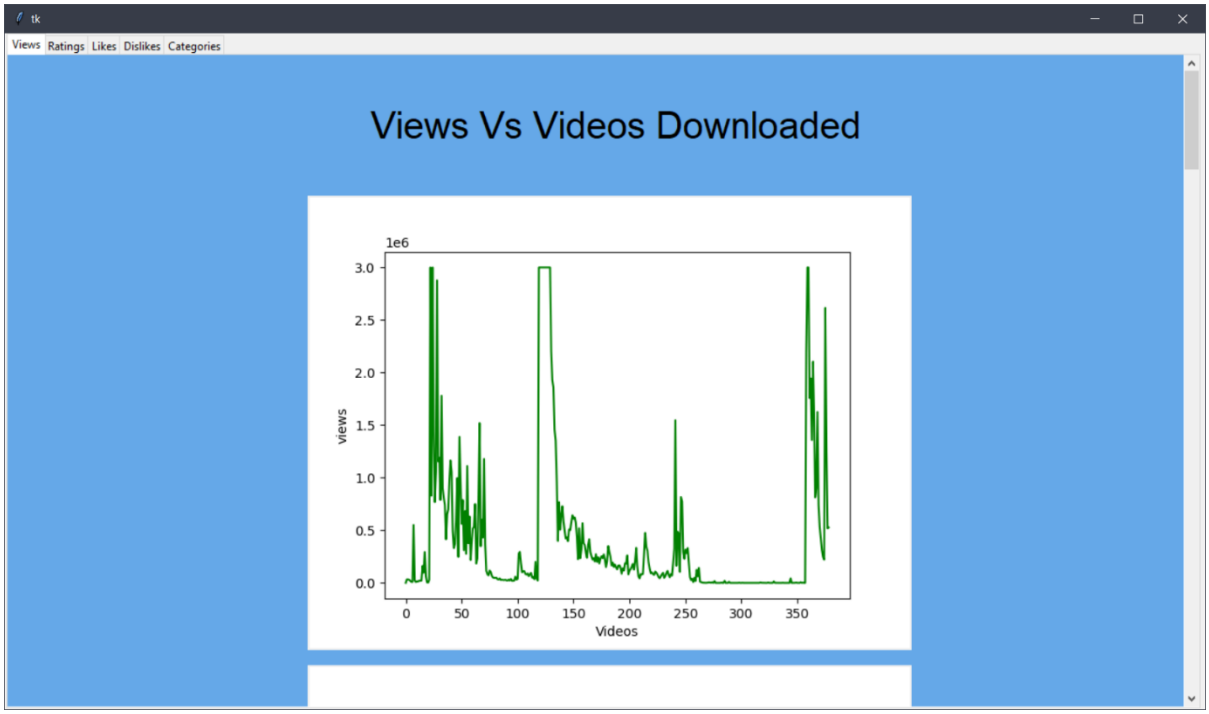

tk
Here's the Playlist!
You can select Multiple files with Ctrl, and remove the ones you don't need!
Kappa
One More!
python
Tkinter
Tkinter Python GUI Tutorial For Beginners 3
Tkinter Python GUI Tutorial For Beginners 3 - Creating First GUI Application with Tkinter
00:015:21
Select Quality :
webm 50Kb/s (only audio)
Change Path :
Select Files to Download :
https://www.youtube.com/watch?v=ls3BAhPV06M
https://www.youtube.com/watch?v=rpgqdR-KHhE
https://www.youtube.com/watch?v=OK_ntOcU28g
https://www.youtube.com/watch?v=bZ6NL59FMoc
https://www.youtube.com/watch?v=eRaO35fvsHs
https://www.youtube.com/watch?v=nIPHwRK51-k
https://www.youtube.com/watch?v=kzp7-0EFrlg
Total Videos : 18
Current No. : 0
Downloaded : 1
Remaining : 18
Skipped : 0
File Size : 4.78 MB
Remove Selected
Path : C:\Users\ittl\Desktop\Programs\Python\IP Project Youtube Video Downloader
Progress :
7.79%
Download

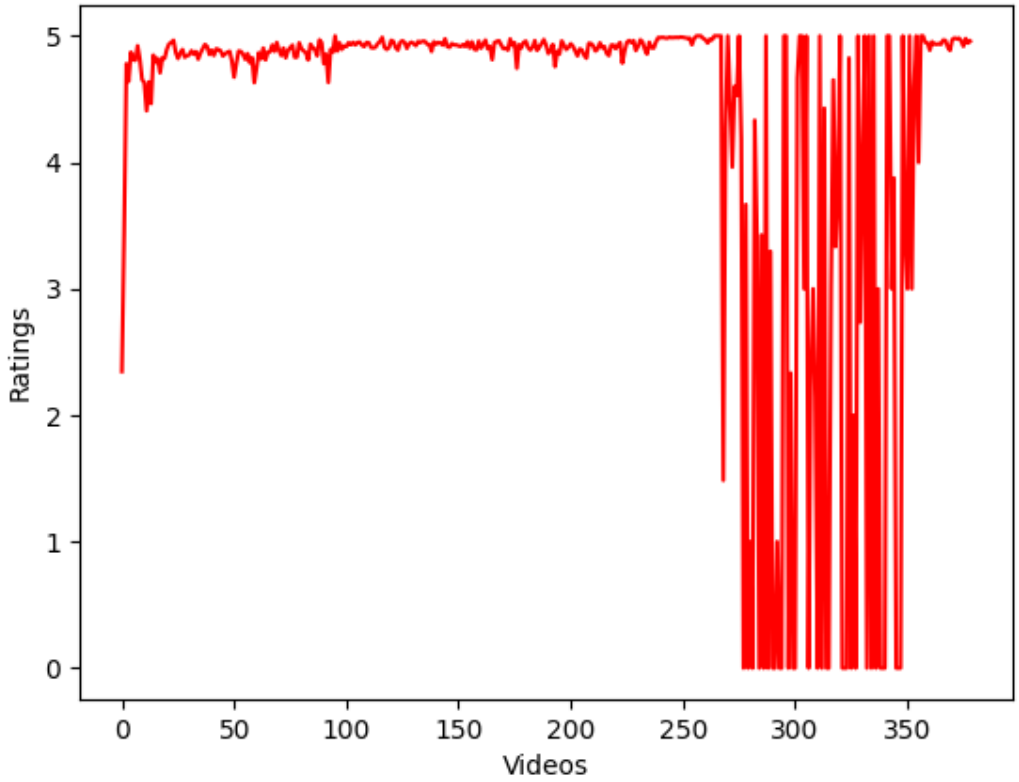

tk
Views Ratings Likes Dislikes Categories
Likes Vs Videos Downloaded
Number of videos vs likes
Number of videos
Views
0
20
40
60
80
100
120
140
160
0
2000
4000
6000
8000
10000
12000
14000

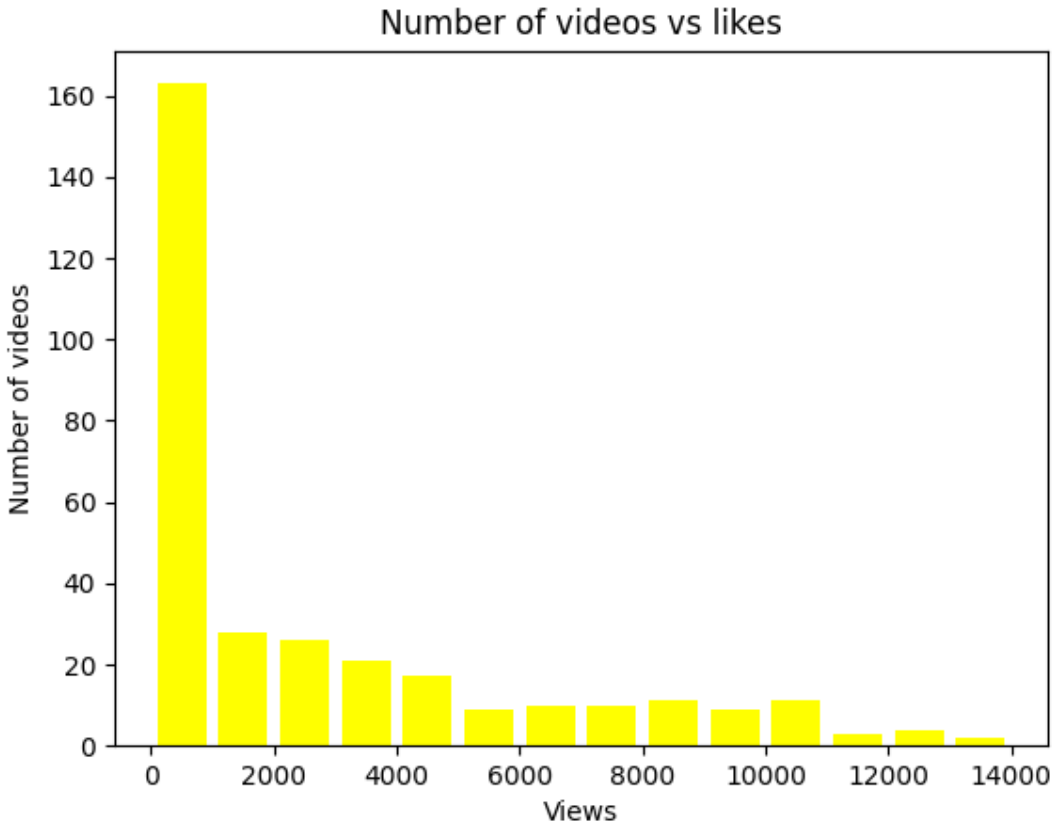

tk
Views Ratings Likes Dislikes Categories
Dislikes Vs Videos Downloaded
dislikes
Videos
0
1000
2000
3000
4000
5000
6000
0
50
100
150
200
250
300
350

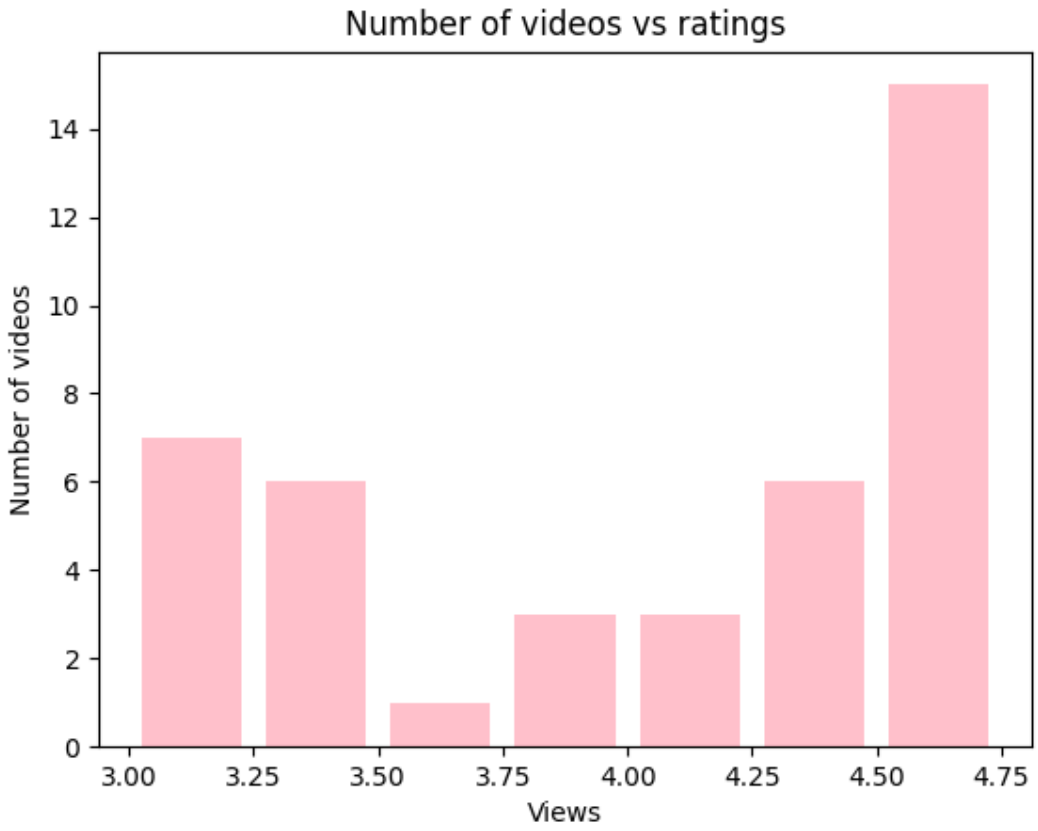

tk
Views
Ratings
Likes
Dislikes
Categories
0
0
50
100
150
200
250
300
350
Videos
Number of videos vs ratings
Number of videos
14
12
10
8
6
4
2
0
3.00
3.25
3.50
3.75
4.00
4.25
4.50
4.75
Views

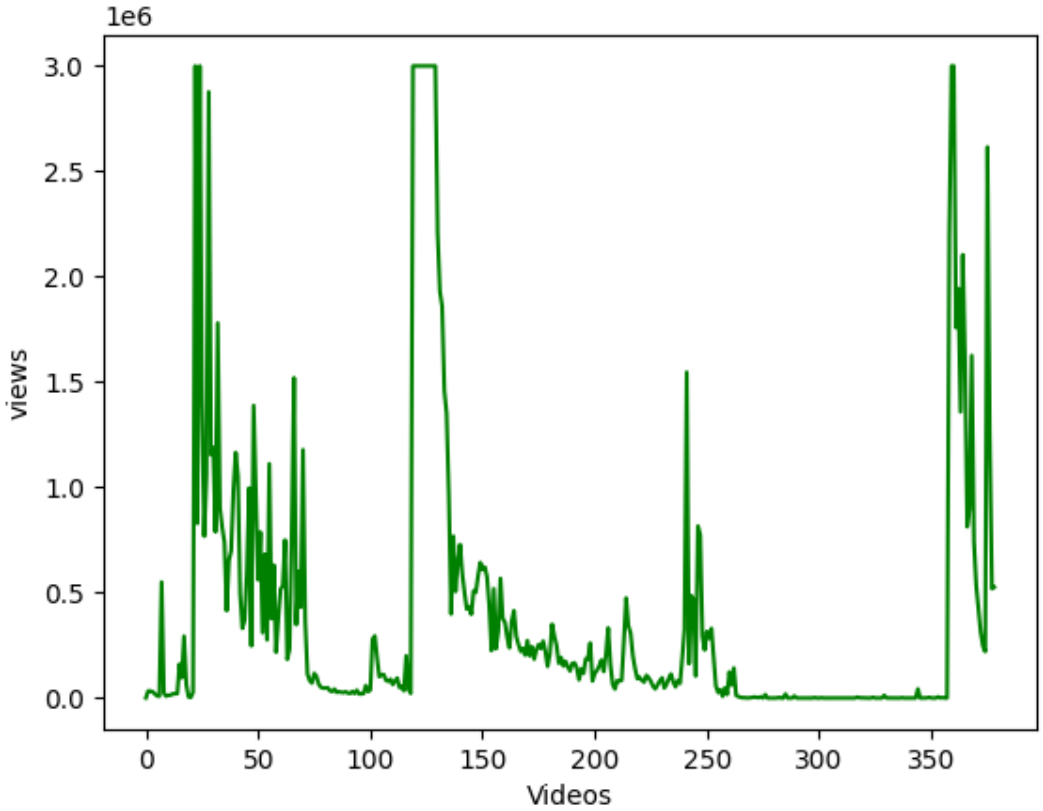

tk
Views
Ratings
Likes
Dislikes
Categories
0
Education
Science
Music
Vehicles
Entertain
How-to
People
Education
Science
People
How-to
Entertain
Music
Vehicles


tk
Views
Ratings
Likes
Dislikes
Categories
Views Vs Videos Downloaded
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
views
0
50
100
150
200
250
300
350
Videos


Ratings
Videos
0
1
2
3
4
5
0
50
100
150
200
250
300
350


Number of videos vs likes
Number of videos
Views
0
20
40
60
80
100
120
140
160
0
2000
4000
6000
8000
10000
12000
14000


Number of videos vs ratings
Number of videos
0
2
4
6
8
10
12
14
3.00
3.25
3.50
3.75
4.00
4.25
4.50
4.75
Views


1e6
views
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
50
100
150
200
250
300
350
Videos

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.