ZIP FILE CONVERTER USING GREEDY ALGORITHM

A PROJECT REPORT

Submitted by

Ritik Vishal (24MCA20196)

in partial fulfillment for the award of the degree of

MASTER OF COMPUTER APPLICATION

IN

COMPUTER SCIENCE

Chandigarh University

APRIL, 2025

TABLE OF CONTENT

Sr no.	Topic	Page No.
1	Problem Statement & Objective	3
2	Introduction	5
3	Set of input output used	6
4	Algorithm and Technique Used	7
5	Constrains	9
6	Code & Screenshot	10
7	References	15

1.Problem Statement

In today's digital age, the exponential growth of data has made efficient file management an essential requirement for both individuals and organizations. With the increasing reliance on digital communication and collaboration, users frequently find themselves dealing with a multitude of files that need to be shared, stored, or organized. This scenario is particularly common in professional environments where teams collaborate on projects, leading to the accumulation of numerous documents, images, and other file types.

However, managing a large number of individual files can be cumbersome and inefficient. Users often face challenges such as:

1. **Complexity of File Management**: As the number of files increases, so does the complexity of managing them. Users may struggle to keep track of which files are relevant, leading to confusion and potential loss of important data. This complexity is exacerbated when files are stored in different locations or folders, making it difficult to locate specific documents quickly.
2. **File Sharing Limitations**: When it comes to sharing files, especially via email or cloud storage, users encounter significant limitations. Many email providers impose strict size limits on attachments, which can hinder the ability to send large files or multiple files at once. Similarly, cloud storage services may have upload restrictions or require users to organize files into specific folders, adding another layer of complexity to the sharing process.
3. **Inefficiency in Data Transfer**: The need to send multiple files individually can lead to inefficiencies in data transfer. Users may spend considerable time zipping files manually or may resort to sending files one at a time, which can be tedious and time-consuming. This inefficiency not only affects productivity but can also lead to frustration among users.
4. **Need for Compression**: Compressing files into a single ZIP archive offers a practical solution to these challenges. By consolidating multiple files into one compressed file, users can significantly reduce the overall file size, making it easier to share and store data. This not only simplifies the process of file sharing but also enhances organization, as users can manage a single ZIP file instead of multiple individual files.

To address these pressing issues, there is a clear need for a solution that empowers users to efficiently manage their files. The proposed solution is a user-friendly application that allows users to easily select multiple files and compress them into a single ZIP archive. This application will streamline the file management process by providing an intuitive interface for file selection, robust error handling to manage potential issues, and clear feedback to users regarding the success or failure of the zipping operation.

By implementing this solution, users will benefit from a more efficient and organized approach to file management. The application will not only simplify the process of sharing files but also enhance productivity by reducing the time and effort required to manage multiple files. Ultimately, this project aims to provide a valuable tool that meets the evolving needs of users in an increasingly digital world.

1.1 Objective

The primary objective of this project is to design and implement a user-friendly graphical user interface (GUI) application that simplifies the process of file management through the use of Python's Tkinter library. The application aims to address the challenges associated with handling multiple files by providing a streamlined solution for file selection and compression. The specific objectives of the project are as follows:

1. **Select Multiple Files**:
   - The application will offer a simple and intuitive interface that allows users to easily select multiple files from their file system. This feature will enable users to navigate their directories and choose the files they wish to compress without any complexity. The goal is to enhance user experience by minimizing the steps required to select files.
2. **Compress Files into a ZIP Archive**:
   - Once the files are selected, the application will provide functionality to compress these files into a single ZIP archive. This feature is crucial for facilitating easier sharing and storage of files, as it reduces the overall file size and consolidates multiple files into one manageable unit. The application will ensure that the compression process is efficient and user-friendly.
3. **Error Handling**:
   - The application will incorporate robust error handling mechanisms to manage various scenarios that may arise during file selection and compression. This includes handling cases where no files are selected, issues with file access (e.g., files not found), or problems encountered during the zipping process. By implementing comprehensive error handling, the application will provide a more reliable and resilient user experience.
4. **User Feedback**:
   - To enhance the overall user experience, the application will provide clear and informative feedback to users regarding the success or failure of the zipping operation. This feedback will be communicated through message boxes or notifications, ensuring that users are aware of the outcome of their actions. By keeping users informed, the application will foster a sense of confidence and satisfaction in its functionality.

By achieving these objectives, the project aims to deliver a valuable tool that simplifies file management for users, making it easier to share and organize files in an increasingly digital environment. The application will not only improve efficiency but also enhance user satisfaction by providing a seamless and intuitive experience. Ultimately, this project seeks to address the growing need for effective file management solutions in both personal and professional contexts.

2. Introduction

In an era where digital data is generated and shared at an unprecedented rate, effective file management has become a critical necessity for both individuals and organizations alike. The proliferation of digital files—ranging from documents and images to videos and presentations—has led to significant challenges in organizing, sharing, and storing these files efficiently. Users often find themselves overwhelmed by the sheer volume of files they need to manage, leading to inefficiencies, confusion, and frustration in their daily tasks.

One common scenario that many users encounter is the need to share multiple files simultaneously. Traditional methods of sharing files, such as sending them individually via email or uploading them to cloud storage, can be cumbersome and time-consuming. Additionally, many email services impose strict size limits on attachments, which can make it difficult to send large files or multiple files at once. This situation not only complicates the sharing process but also highlights the need for a more streamlined approach to file management that simplifies the organization and transfer of files.

To address these pressing challenges, this project aims to develop a user-friendly graphical user interface (GUI) application using Python's Tkinter library. The application, referred to as "File Zipper," will empower users to easily select multiple files from their file system and compress them into a single ZIP archive. By consolidating multiple files into one compressed file, users can significantly reduce the overall file size, making it easier to share and store their data without the hassle of dealing with numerous individual files.

The application will not only facilitate file compression but will also incorporate robust error handling to manage potential issues, such as no files being selected or problems encountered during the zipping process. Furthermore, it will provide clear and informative feedback to users regarding the success or failure of their actions, thereby enhancing the overall user experience and ensuring that users feel confident in their interactions with the application.

In summary, the "File Zipper" application seeks to simplify the process of file management by providing an intuitive and efficient solution for compressing and sharing files. By addressing the common challenges associated with handling multiple files, this project aims to improve productivity and user satisfaction in an increasingly digital world, ultimately contributing to a more organized and efficient approach to file management for users across various contexts.

3.Input and Output

In the "File Zipper" application, the interaction between the user and the software involves specific inputs and outputs that facilitate the file compression process. Below is a detailed description of the inputs and outputs used in the application:

Inputs

1. **File Selection**:
    - **Input Method**: The user initiates the file selection process by clicking the "Select Files" button.
    - **Input Type**: A file dialog opens, allowing the user to browse their file system and select multiple files.
    - **Expected Input**: The user can select one or more files of any type (e.g., documents, images, videos) to be included in the ZIP archive.
2. **ZIP File Name**:
    - **Input Method**: After selecting the files, the user clicks the "Zip Files" button.
    - **Input Type**: A save dialog opens, prompting the user to specify the name and location for the ZIP file.
    - **Expected Input**: The user must provide a valid file name for the ZIP archive, which will have a ".zip" extension.

Outputs

1. **Display of Selected Files**:
    - **Output Method**: The application displays the list of selected files in a designated area of the GUI.
    - **Output Type**: A text area shows the full paths of the selected files, formatted as a list.
    - **Expected Output**: The user can visually confirm which files they have selected for compression.
2. **Success Message**:
    - **Output Method**: Upon successful creation of the ZIP file, a message box appears.
    - **Output Type**: A pop-up notification informs the user that the files have been successfully zipped.
    - **Expected Output**: The message will read: "Files successfully zipped into [ZIP file path]."
3. **Error Messages**:
    - **Output Method**: If an error occurs during file selection or zipping, an error message box is displayed.
    - **Output Type**: A pop-up notification alerts the user to the specific issue encountered.
    - **Expected Output**: Possible error messages include:
        - "Error: No files selected!" (if the user attempts to zip without selecting any files)
        - "Error: An error occurred: [error details]" (if there is an issue during the zipping process, such as file access problems)

4. Algorithm and Techniques Used

The "File Zipper" application employs a straightforward algorithmic approach to achieve its functionality of selecting and compressing files into a ZIP archive. Below is a detailed explanation of the algorithm and techniques used in the implementation:

1. Graphical User Interface (GUI) Design

The application is built using Python's Tkinter library, which provides a robust framework for creating graphical user interfaces. The GUI design follows a simple and intuitive layout, allowing users to interact with the application easily. The key components of the GUI include:

- **File Selection Button**: This button triggers a file dialog that allows users to select multiple files from their file system.
- **Display Area**: A label displays the list of selected files, providing users with visual confirmation of their selections.
- **Zip Files Button**: This button initiates the compression process, prompting users to specify the name and location for the ZIP file.

2. File Selection Algorithm

The file selection process is handled by the **select_files()** function, which utilizes the following steps:

- **Open File Dialog**: The function calls **filedialog.askopenfilenames()**, which opens a dialog for users to select multiple files. This function returns a tuple of file paths selected by the user.
- **Update Display**: The selected file paths are then joined into a single string with newline characters and set to the **file_list** variable, which updates the display area in the GUI.

3. File Compression Algorithm

The compression of selected files into a ZIP archive is managed by the **zip_selected_files()** function, which follows these steps:

- **Retrieve Selected Files**: The function retrieves the list of selected files from the **file_list** variable and splits it into individual file paths.
- **Error Handling**: Before proceeding, the function checks if any files were selected. If not, it displays an error message using **messagebox.showerror()**.
- **Open Save Dialog**: The function calls **filedialog.asksaveasfilename()** to prompt the user to specify the name and location for the ZIP file. This function ensures that the user provides a valid file name with a ".zip" extension.
- **Create ZIP Archive**: Using the **zipfile.ZipFile** class, the function creates a new ZIP file in write mode. It iterates through the list of selected files, checking if each file exists. If a file exists, it is added to the ZIP archive using the **write()** method, which also allows for specifying the file name within the archive.
- **Success and Error Notifications**: Upon successful completion of the zipping process, a success message is displayed using **messagebox.showinfo()**. If any exceptions occur during the process (e.g., file access issues), an error message is shown to the user.

4. Error Handling Techniques

The application incorporates robust error handling techniques to manage potential issues that may arise during file selection and compression. This includes:

- **Input Validation**: The application checks if any files were selected before attempting to create a ZIP file. If no files are selected, an error message is displayed.
- **Exception Handling**: The use of a try-except block allows the application to catch and handle exceptions that may occur during the zipping process, providing users with informative error messages.

5. Constraints

While developing the "File Zipper" application, several constraints were identified that could impact its functionality, usability, and performance. Understanding these constraints is essential for evaluating the application's effectiveness and for planning future enhancements. Below are the key constraints associated with the project:

1. File Type Limitations

- The application allows users to select any type of file for compression. However, certain file types (e.g., system files, hidden files) may not be accessible or may require special permissions to be included in the ZIP archive. This could limit the usability of the application in specific scenarios.

2. File Size Limitations

- While the application can handle multiple files, the total size of the files being compressed may be constrained by the available system memory and disk space. If users attempt to compress a very large number of files or particularly large files, they may encounter performance issues or errors related to insufficient resources.

3. Operating System Compatibility

- The application is developed using Python and Tkinter, which are generally cross-platform. However, certain functionalities, such as file dialogs and file path handling, may behave differently across operating systems (e.g., Windows, macOS, Linux). This could lead to inconsistencies in user experience depending on the platform used.

4. User Interface Limitations

- The GUI is designed to be simple and intuitive, but it may not accommodate all user preferences or accessibility needs. For example, users with visual impairments may find it challenging to interact with the application if it lacks features such as screen reader compatibility or customizable font sizes.

5. Error Handling Constraints

- While the application includes basic error handling, it may not cover all possible error scenarios. For instance, if a user selects files that are in use or locked by another process, the application may not provide specific feedback on the nature of the error. This could lead to user confusion and frustration.

6. Dependency on External Libraries

- The application relies on the **zipfile** module from Python's standard library and the Tkinter library for GUI functionality. Any issues or bugs in these libraries could affect the performance and reliability of the application. Additionally, users must have Python installed on their systems to run the application.

7. Network Dependency for Cloud Storage

- If users intend to upload the resulting ZIP file to cloud storage or share it via email, the application does not include built-in functionality for these actions. Users must rely on external applications or services, which may introduce additional constraints related to network connectivity and file size limits imposed by those services.

6. Code & Screenshots

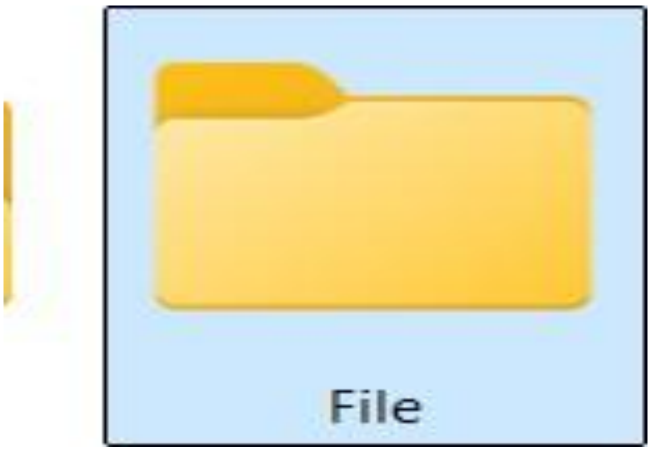
6.1. Code

```
import os
import zipfile
import tkinter as tk
from tkinter import filedialog, messagebox

def select_files():
    """Open a file dialog to select multiple files"""
    files = filedialog.askopenfilenames(title="Select Files to Zip")
    file_list.set("\n".join(files))  # Display selected files in the GUI
    return files

def zip_selected_files():
    """Compress the selected files into a ZIP archive"""
    files = file_list.get().split("\n")

    if not files or files == [""]:
        messagebox.showerror("Error", "No files selected!")
        return

    zip_filename = filedialog.asksaveasfilename(defaultextension=".zip",
                                                filetypes=[("ZIP Files", "*.zip")],
                                                title="Save ZIP File")

    if not zip_filename:
        return  # User canceled save dialog

    try:
        with zipfile.ZipFile(zip_filename, 'w') as zipf:
            for file in files:
                if os.path.exists(file):
                    zipf.write(file, os.path.basename(file))  # Add file to ZIP
        messagebox.showinfo("Success", f"Files successfully zipped into {zip_filename}")
    except Exception as e:
        messagebox.showerror("Error", f"An error occurred: {e}")

# Create GUI Window
root = tk.Tk()
root.title("File Zipper")
root.geometry("500x400")

# UI Elements
file_list = tk.StringVar()
```

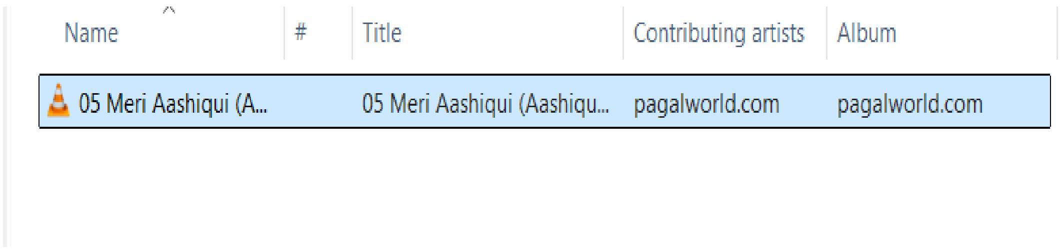
```
tk.Label(root, text="Selected Files:", font=("Arial", 12)).pack(pady=10)
file_display = tk.Label(root, textvariable=file_list, bg="white", relief="solid", width=60, height=10,
anchor="nw", justify="left")
file_display.pack(pady=5)

tk.Button(root, text="Select Files", command=select_files, font=("Arial", 12),
bg="lightblue").pack(pady=10)
tk.Button(root, text="Zip Files", command=zip_selected_files, font=("Arial", 12),
bg="lightgreen").pack(pady=10)

# Run the GUI
root.mainloop()
```

6.2. Screenshot

a. Create a new empty file

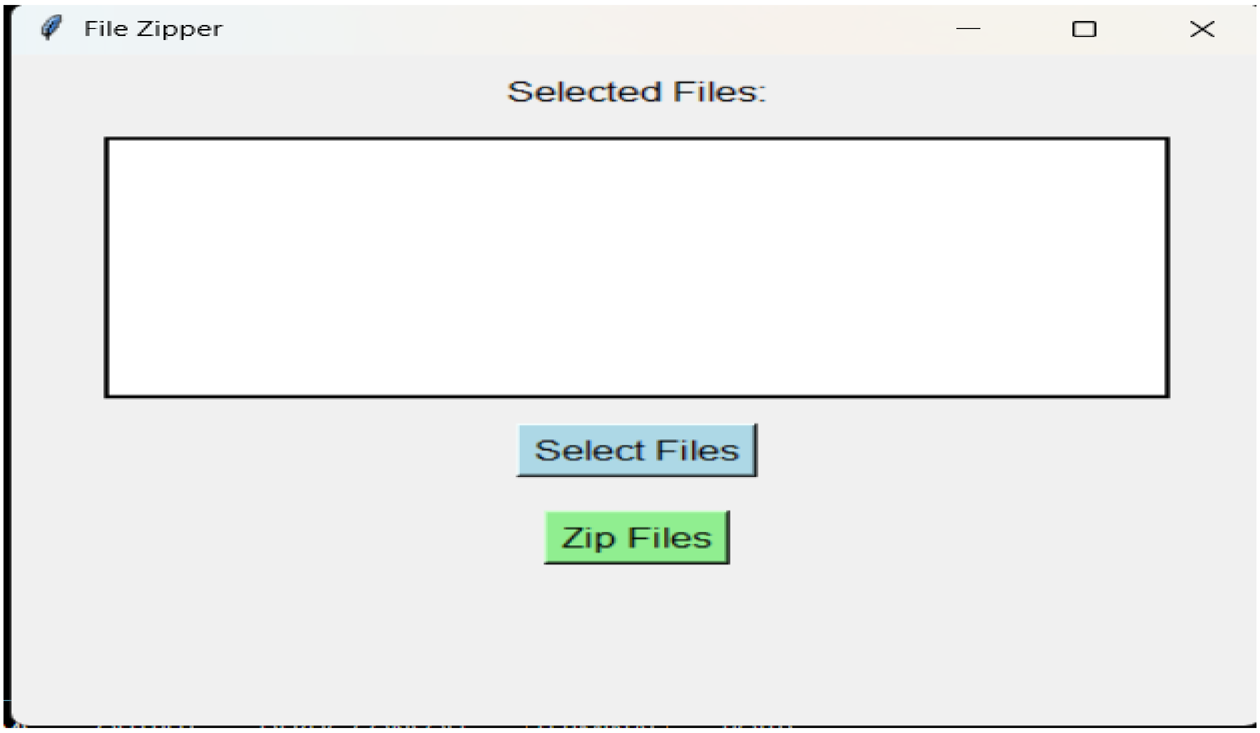


b. Add item in empty file



Name	#	Title	Contributing artists	Album
05 Meri Aashiqui (A...		05 Meri Aashiqui (Aashiqu...	pagalworld.com	pagalworld.com

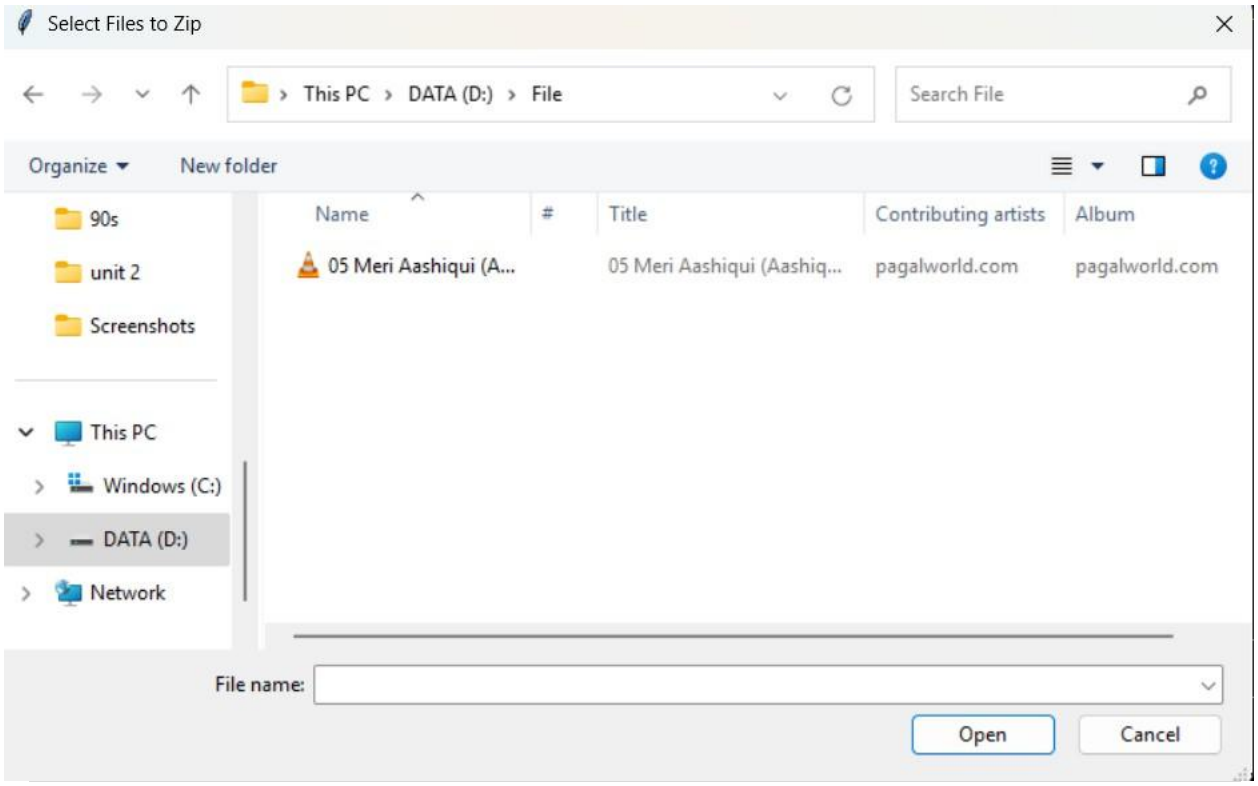
c. **Run the program**



d. **Select the file**

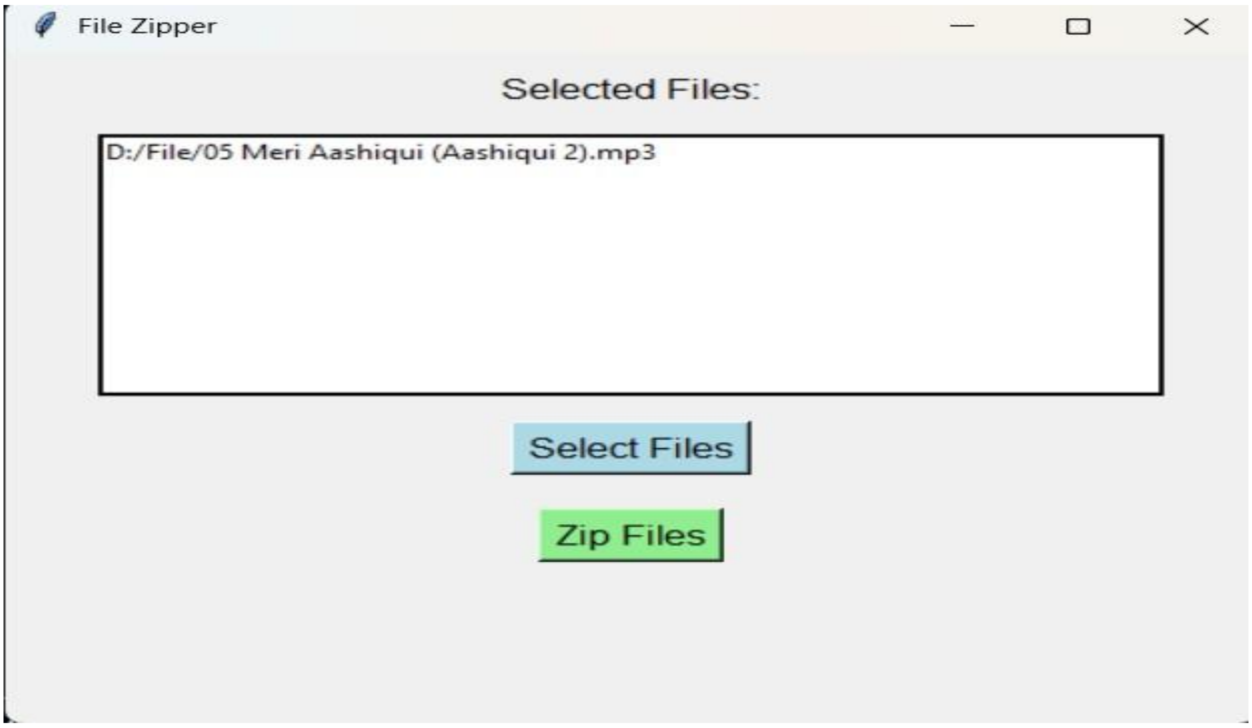
File 30-03-2025 09:43 AM File folder

e. **Select the item**



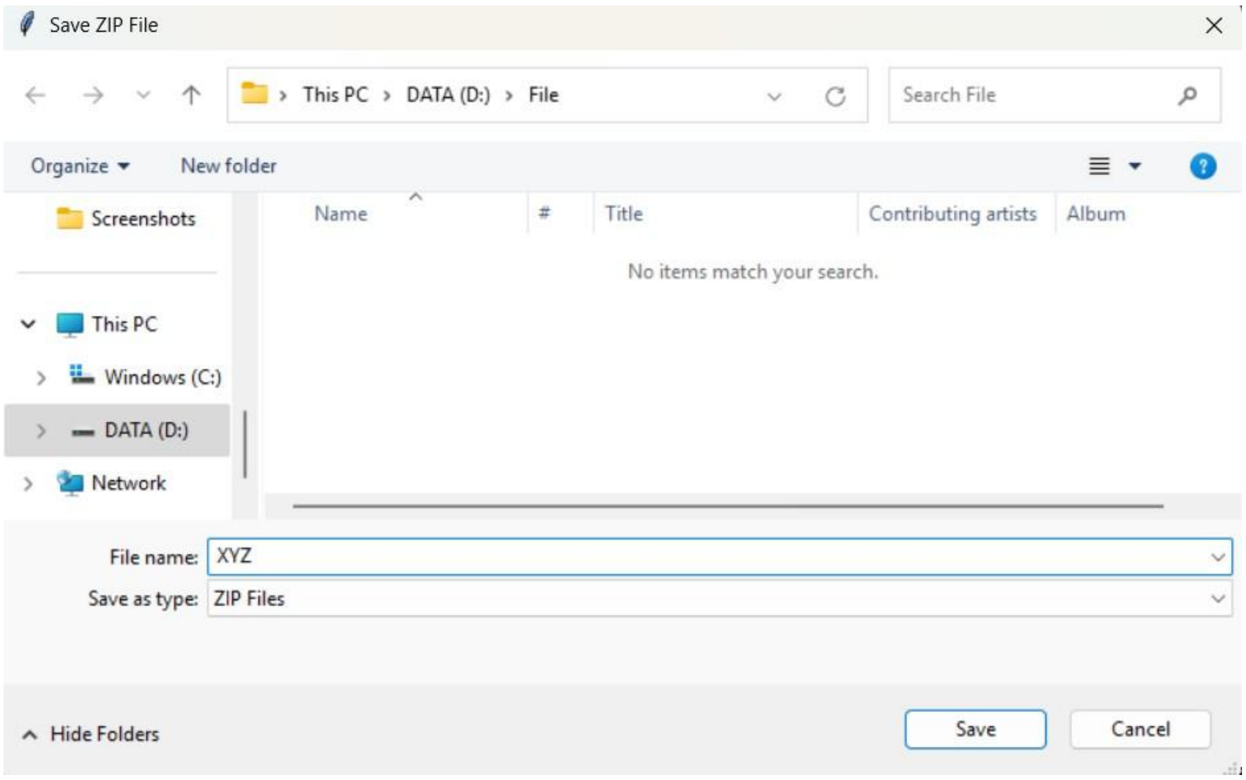
Name	#	Title	Contributing artists	Album
05 Meri Aashiqui (A...		05 Meri Aashiqui (Aashiq...	pagalworld.com	pagalworld.com

f. <u>Open this Item</u>



D:/File/05 Meri Aashiqui (Aashiqui 2).mp3

g. <u>Give the tittle of File</u>

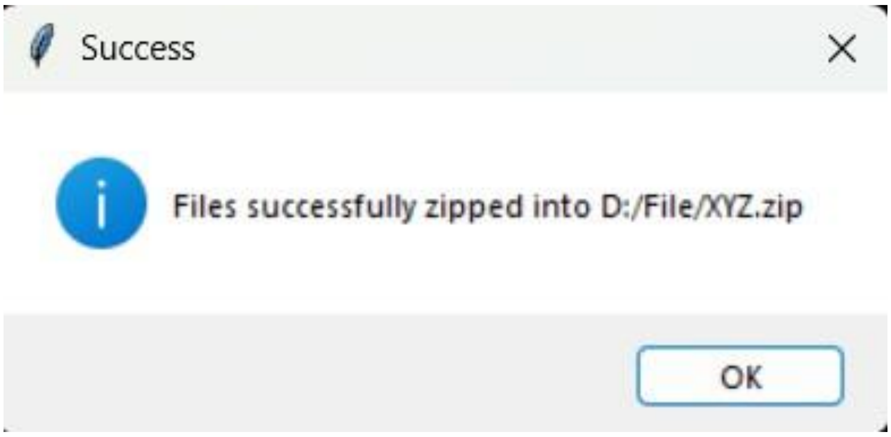


No items match your search.

File name: XYZ

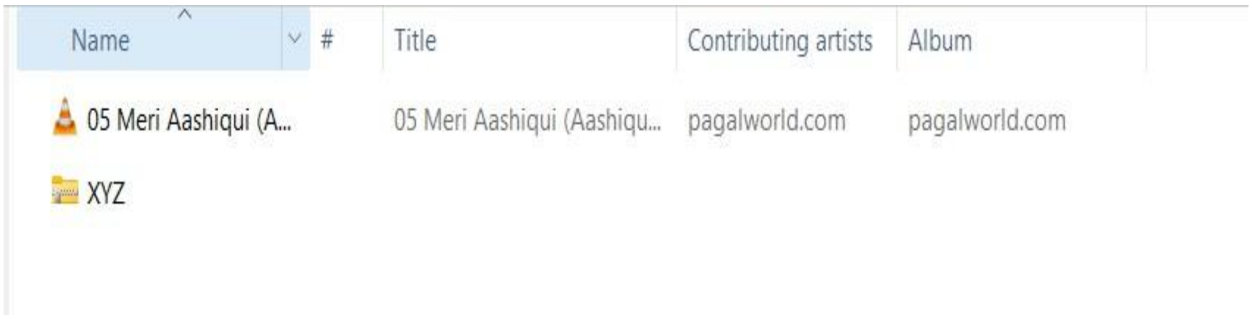
Save as type: ZIP Files

h. **Then click on Zip Files button**



Files successfully zipped into D:/File/XYZ.zip

i. **Ready to share Zip File**



Name	#	Title	Contributing artists	Album
05 Meri Aashiqui (A...		05 Meri Aashiqui (Aashiqu...	pagalworld.com	pagalworld.com
XYZ				

7. References

1. Python Software Foundation. (2023). *Python Documentation*. Retrieved from https://docs.python.org/3/
2. Tkinter Documentation. (2023). *Tkinter — Python interface to Tcl/Tk*. Retrieved from https://docs.python.org/3/library/tkinter.html
3. Python Software Foundation. (2023). *zipfile — Work with ZIP archives*. Retrieved from https://docs.python.org/3/library/zipfile.html
4. Lutz, M. (2013). *Learning Python*. O'Reilly Media.
5. Grayson, J. (2000). *Python and Tkinter Programming*. Manning Publications.
6. Beazley, D. M. (2009). *Python Essential Reference*. Addison-Wesley.
7. Sweigart, A. (2015). *Automate the Boring Stuff with Python*. No Starch Press.
8. Python GUI Programming with Tkinter. (2021). *Real Python*. Retrieved from https://realpython.com/python-gui-tkinter/
9. W3Schools. (2023). *Python File Handling*. Retrieved from https://www.w3schools.com/python/python_file_handling.asp
10. GeeksforGeeks. (2023). *Python | Zipfile module*. Retrieved from https://www.geeksforgeeks.org/python-zipfile-module/
11. Python GUI Programming Cookbook. (2018). Packt Publishing.
12. Tkinter GUI Application Development Blueprints. (2018). Packt Publishing.
13. Python Programming and Numerical Methods: A Guide for Engineers and Scientists. (2019). Springer.
14. Kumar, A. (2020). *Python GUI Programming with Tkinter*. Packt Publishing.
15. Python for Data Analysis. (2017). O'Reilly Media.
16. Python Crash Course: A Hands-On, Project-Based Introduction to Programming. (2019). No Starch Press.
17. Zelle, J. (2010). *Python Programming: An Introduction to Computer Science*. Franklin, Beedle & Associates Inc.
18. Tkinter: The Definitive Guide. (2020). O'Reilly Media.
19. Python Programming: An Introduction to Computer Science. (2016). Franklin, Beedle & Associates Inc.
20. Real Python. (2023). *Working with Files in Python*. Retrieved from https://realpython.com/python-files/