Degree Project:

File Handling in Python



Björn Lagerblad

NBI/Handelsakademin

File Handling in Python

2025/02

# 

# Abstract

I chose to write this essay in English so that any future employers or curious visitors to my GitHub can easily understand its purpose.

This project focuses on the development of a Python script to automate file organization using **pathlib, shutil** and **tkinter**.

The script identifies files in a given directory, categorizes them based on file type, and moves them into designated folders for better organization. This automation reduces manual effort and improves overall efficiency in managing files. The script was tested across multiple environments, including Windows, Linux, and Raspberry Pi, ensuring cross-platform compatibility. Future improvements include the addition of a graphical user interface (GUI) to further enhance user experience.

To ensure clarity and correctness, I used ChatGPT to proofread and refine the English wording of this report, as I’m eager to explore new tools and I see it as an excellent way to test the LLM/Human integration of language barriers and terminology.

# Förkortningar och Begrepp

**Skapas automatiskt i Word genom att gå till Referenser > Innehållsförteckning.**

Table of Contents

[1 Abstract 2](#_Toc191406224)

[2 Förkortningar och Begrepp 3](#_Toc191406225)

[3 Introduction 1](#_Toc191406226)

[4 Theory 2](#_Toc191406227)

[4.1 Cross-Platform File Systems 2](#_Toc191406228)

[4.2 Python’s File Management Approach 2](#_Toc191406229)

[4.3 GUI Theory and Principles 3](#_Toc191406230)

[4.4 Error Handling & Robustness 3](#_Toc191406231)

[4.5 Automation & Productivity 3](#_Toc191406232)

[4.6 Summary 4](#_Toc191406233)

[2 Metod 5](#_Toc191406234)

[3 Resultat och Diskussion 6](#_Toc191406235)

[4 Slutsatser 7](#_Toc191406236)

[Appendix A 8](#_Toc191406237)

[Källförteckning 9](#_Toc191406238)

# Introduction

I found it prudent to create something simpler yet with a new challenge: a file organizer that works across different operating systems and includes a functional GUI.

During larger projects, I noticed that many of my classmates have massive download folders with scattered files, resulting in an unorganized mess that makes efficiency worse.

Automating this process can (hopefully) help future students keep their files in order, and it will also serve as a good example of thorough code documentation.

This project focuses on developing a Python script to automate file organization. The main goals of the script are to:

* Identify files in a folder
* Sort them based on file type (e.g., images, documents, videos)
* Move them to specific directories for better organization
* Implement a working GUI with robust error handling
* Ensure compatibility across multiple operating systems

The script is built using **Python 3.11** and its standard libraries, particularly **pathlib** for file handling. If time permits, a **Graphical User Interface (GUI)** using **tkinter** or another Python framework will be added to improve usability.

Ultimately, the aim is to challenge myself with something new (GUI handling in this case) in Python while documenting every step of the process, from planning and coding to testing and analysis.

# Theory

In this section, I’ll explain the main ideas behind how this project will work. First, we’ll look at how different operating systems handle files and why that matters between the different Operating Systems today. Then, I’ll discuss Python’s built-in tools—like **pathlib** and **shutil**—that works such as moving, copying, and deleting files without having to worry about path separators or manual string manipulation. Finally, I’ll give a quick overview of how a GUI (graphical user interface) deals with user interactions, making the program more accessible for people who don’t know terminal commands.

## Cross Platform File Systems

Operating systems like Windows, Linux, and macOS (to name the most popular ones) differ in how they manage file paths, directory structures, and permissions some examples are.

* **Windows:** Commonly uses backslashes(\) in file paths (e.g. C:\Users\Documents), and file names are often case-insensitive.
* **Linux/macOS:** Use forward slashes (/) in file paths (e.g., /home/user/Documents), and typically have case-sensitive file systems.
* **Permissions:** Each system has its own approach to user permissions, read/write access, and hidden files.

For a file organizer to be truly cross-platform, it must handle these variables perfect, or we will get more errors than a new computer in a senior home. By researching libraries and patterns that works away these differences, we can find that some libraries in python will function with that goal in mind.

## Python’s File Management Approach

Python provides multiple ways to interact with files and directories. Historically, the os module (and os path) was the standard tool, meaning you had to manually handle paths in your code. However, modern Python has introduced more user-friendly options that simplify these tasks.:

* **pathlib:**  
  Offers an object-oriented approach to file paths, simplifying code and improving readability. It automatically manages path separators, so the same code works on Windows, Linux, and macOS without manual string manipulation. A great tool with our cross platform goal in mind.
* **shutil:**  
  Provides high-level file operations such as copying, moving, and deleting directories. It complements pathlib by handling tasks that go beyond simple path manipulation, aka we have the tool that works in tandem with pathlib.

## GUI Theory and Principles

A Graphical User Interface (GUI) allows users to interact with an application through visual elements like buttons, menus, and dialog boxes, rather than the command line. In Python, **tkinter** is a common library for building simple GUIs:

* **The User Interface:**
* GUIs rely on events (e.g buttons, key presses and so on from user input) something I have not worked on before and i am approaching this challenge with an open mind.
* **User-Friendliness:**  
  A GUI needs to be as straightforward as possible, so I plan to test it with friends who have minimal technical experience some who admit they struggle even with microwaves. If they can navigate the interface easily, then it’s truly user-friendly.

## Error Handling & Robustness

When working with file operations, robust error handling is essential to prevent data loss or application crashes. Potential issues include:

* **Permission Errors:**  
  Attempts to move or delete files without the necessary read/write permissions.
* **Locked Files:**  
  Some files may be open in other programs, preventing modifications.
* **Invalid Paths:**  
  Missing or broken directory references, especially on systems where a path may not exist or has changed.
* **Different OS Handling:**  
  Different Operating Systems have different sets of rules for file permissions, hidden files and so on, I think this will be the biggest hurdle to get over.

By using exception handling and meaningful error messages, I can help users from clearly know what the issue is so they know what the problem is.

## Automation & Productivity

Manual file sorting can be tedious and prone to human error, especially in environments with high file turnover (e.g. large download folders for teaching a neural network). Automating these processes will in turn:

* **Save Time:**  
  Routinely using this program will save a lot of headaches in not needing to manual sort the files.
* **Reduces Errors:**  
  Consistently using the same script will decrease human error and minimize misplaced filkes.
* **Improves Organization:**  
  Folders and subfolders stay orderly, making it easier to locate and manage files later.

## Summary

By learning about cross-platform file systems, exploring Python’s powerful libraries, and planning a user-friendly GUI with a rudimentary error handling, I’ve laid out a clear foundation for this file organizer. The concepts I covered ranging from path abstractions with pathlib and shutil to the benefits of automation and with what I need to have in mind starting the project.

# Metod

Hur har du genomfört ditt arbete? Exempelvis, hur har datan erhållits?

# Resultat och Diskussion

|  |  |
| --- | --- |
| **RMSE för olika modeller** | |
| Enkel Linjär Regression | xx |
| Lasso | xx |
| Ridge | xx |

Tabell 1: Root Mean Squared Error (RMSE) för de fyra valda modellerna.

En bild som visar text, Teckensnitt, skärmbild, linje

Automatiskt genererad beskrivning

Figur 1: Hur man lägger in tabell eller figur nummer samt beskrivning.

# Slutsatser

Här besvarar du frågeställningarna.

# Appendix A

# Källförteckning