[Your College Logo]

Industrial Internship Report on

“ File Organizer using Python ”

Prepared by

KUMAR PRATHMESH

Executive Summary
This report provides details of the Industrial Internship provided by upskill Campus and The IoT Academy in collaboration with Industrial Partner UniConverge Technologies Pvt Ltd (UCT).   This internship was focused on a project/problem statement provided by UCT. We had to finish the project including the report in 6 weeks’ time.   My project was (Tell about ur Project)   This internship gave me a very good opportunity to get exposure to Industrial problems and design/implement solution for that. It was an overall great experience to have this internship.

TABLE OF CONTENTS

1 Preface 3
2 Introduction 4
2.1 About UniConverge Technologies Pvt Ltd 4
2.2 About upskill Campus 8
2.3 Objective 9
2.4 Reference 9
2.5 Glossary 10
3 Problem Statement 11
4 Existing and Proposed solution 12
5 Proposed Design/ Model 13
5.1 High Level Diagram (if applicable) 13
5.2 Low Level Diagram (if applicable) 13
5.3 Interfaces (if applicable) 13
6 Performance Test 14
6.1 Test Plan/ Test Cases 14
6.2 Test Procedure 14
6.3 Performance Outcome 14
7 My learnings 15
8 Future work scope 16

1 Preface

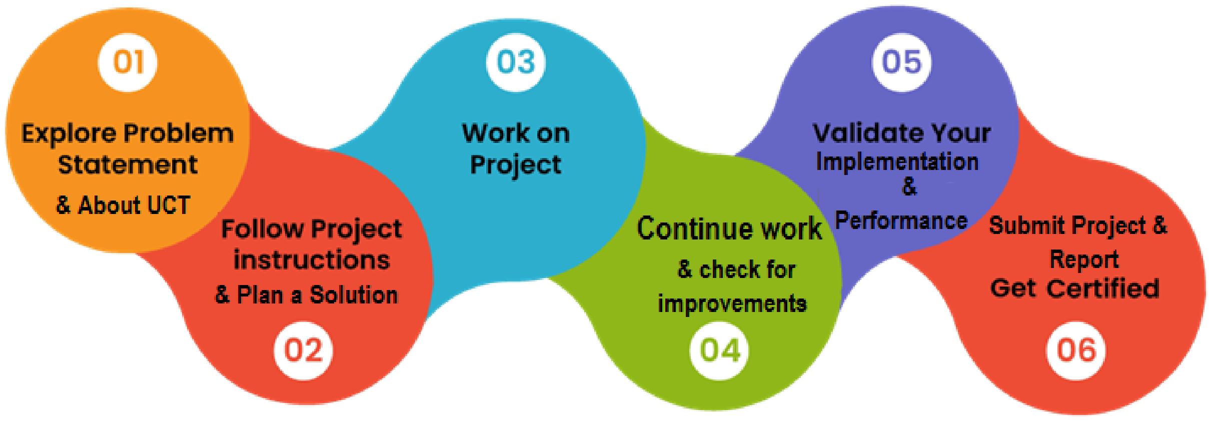
Summary of the whole 6 weeks' work.

About need of relevant Internship in career development.

Brief about Your project/problem statement.

Opportunity given by USC/UCT.

How Program was planned



Your Learnings and overall experience.

Thank to all (with names), who have helped you directly or indirectly.

Your message to your juniors and peers.

2 Introduction

2.1 About UniConverge Technologies Pvt Ltd

A company established in 2013 and working in Digital Transformation domain and providing Industrial solutions with prime focus on sustainability and RoI.

For developing its products and solutions it is leveraging various **Cutting Edge Technologies e.g. Internet of Things (IoT), Cyber Security, Cloud computing (AWS, Azure), Machine Learning, Communication Technologies (4G/5G/LoRaWAN), Java Full Stack, Python, Front end** etc.

uct
Uniconverge Technologies
IIOT Products
We offer product ranging from Remote IOs, Wireless IOs, LoRaWAN Sensor Nodes/ Gateways, Signal converter and IoT gateways
IIOT Solutions
We offer solutions like OEE, Predictive Maintenance, LoRaWAN based Remote Monitoring, IoT Platform, Business Intelligence…
OEM Services
We offer solutions ranging from product design to final production we handle everything for you..

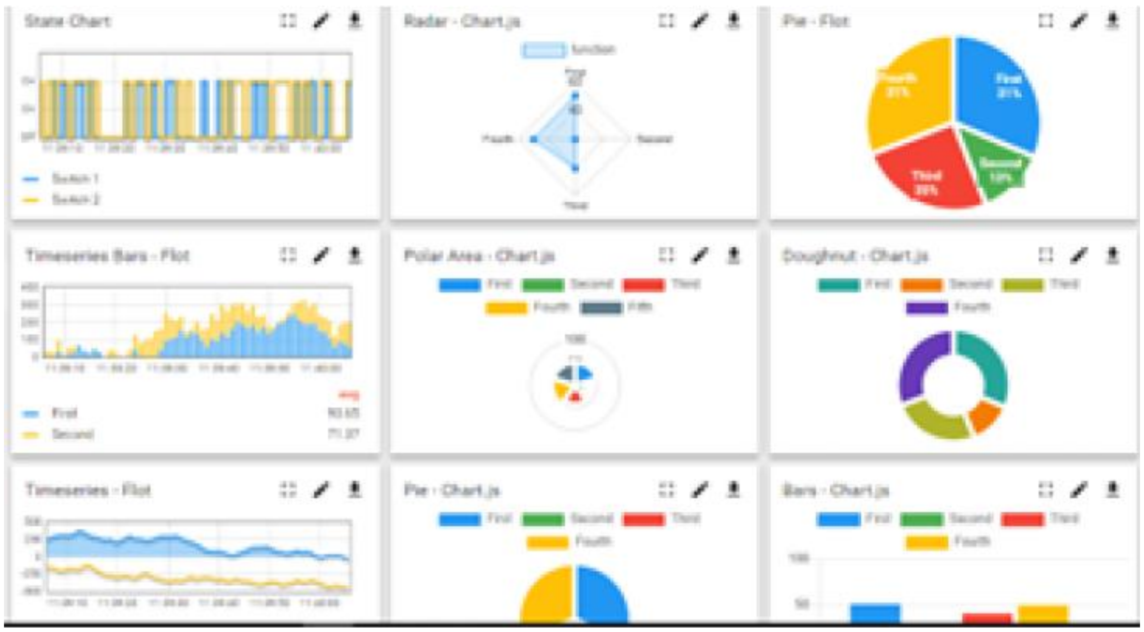
i. UCT IoT Platform (uct Insight)

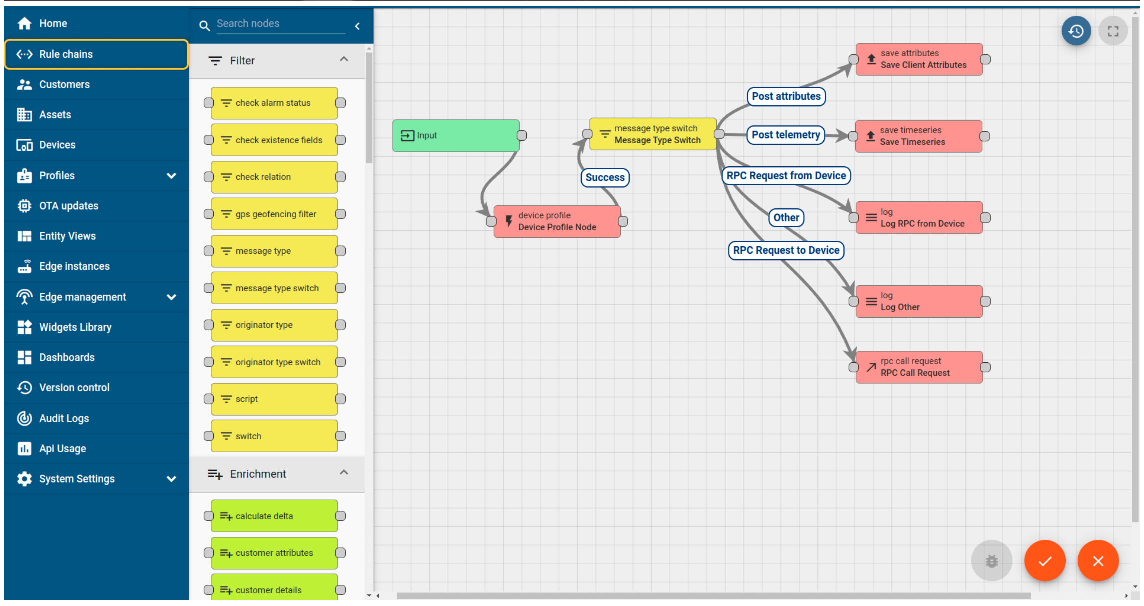
UCT Insight is an IOT platform designed for quick deployment of IOT applications on the same time providing valuable "insight" for your process/business. It has been built in Java for backend and ReactJS for Front end. It has support for MySQL and various NoSql Databases.

- It enables device connectivity via industry standard IoT protocols - MQTT, CoAP, HTTP, Modbus TCP, OPC UA
- It supports both cloud and on-premises deployments.

It has features to

- Build Your own dashboard
- Analytics and Reporting
- Alert and Notification
- Integration with third party application(Power BI, SAP, ERP)
- Rule Engine





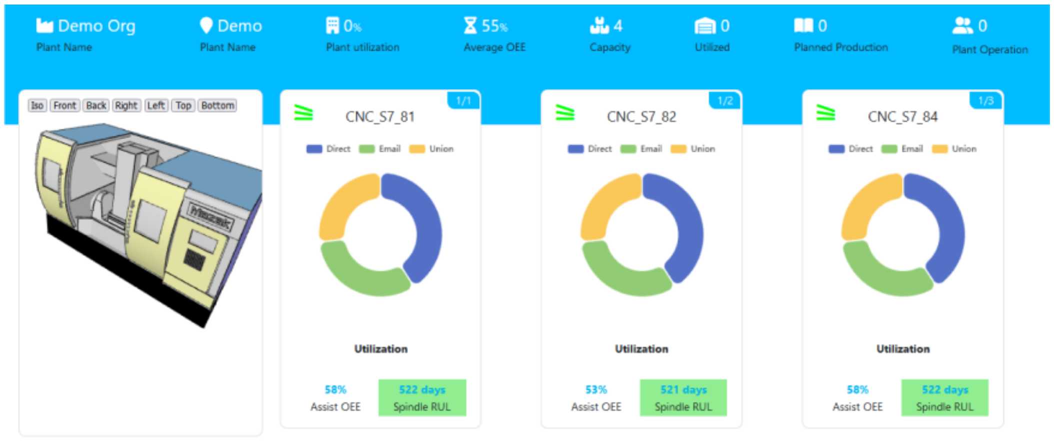
ii. Smart Factory Platform (FACTORY WATCH)

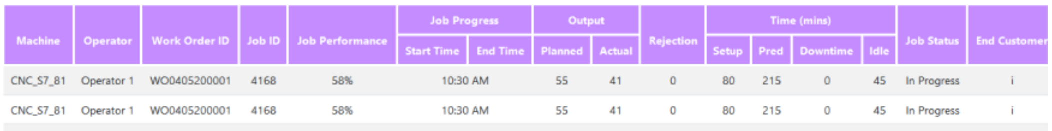
Factory watch is a platform for smart factory needs.

It provides Users/ Factory

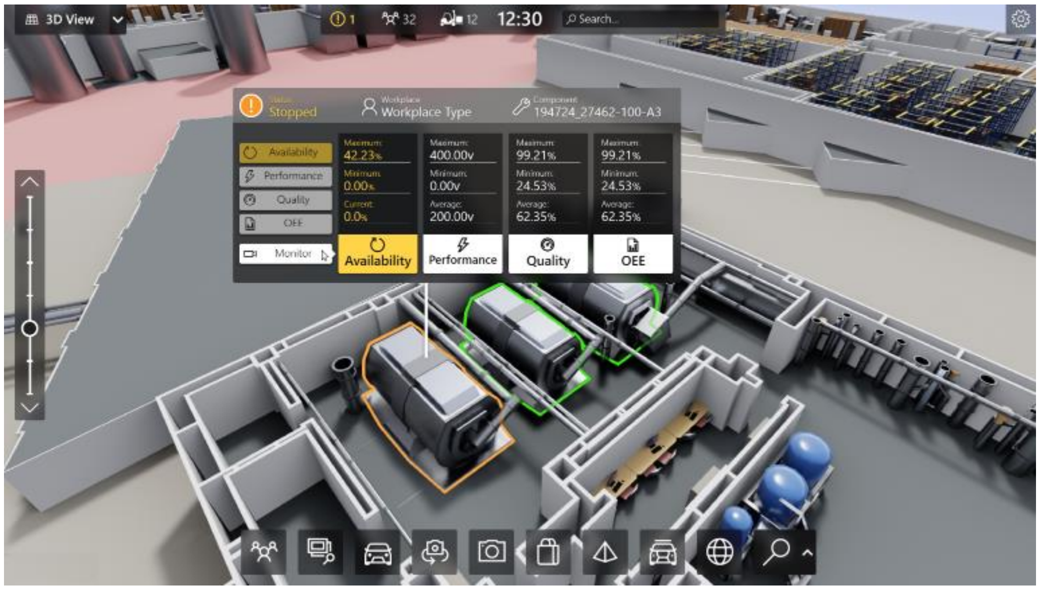
- with a scalable solution for their Production and asset monitoring
- OEE and predictive maintenance solution scaling up to digital twin for your assets.
- to unleased the true potential of the data that their machines are generating and helps to identify the KPIs and also improve them.
- A modular architecture that allows users to choose the service that they what to start and then can scale to more complex solutions as per their demands.

Its unique SaaS model helps users to save time, cost and money.





Machine	Operator	Work Order ID	Job ID	Job Performance	Job Progress		Output		Rejection	Time (mins)				Job Status	End Customer
					Start Time	End Time	Planned	Actual		Setup	Pred	Downtime	Idle		
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i

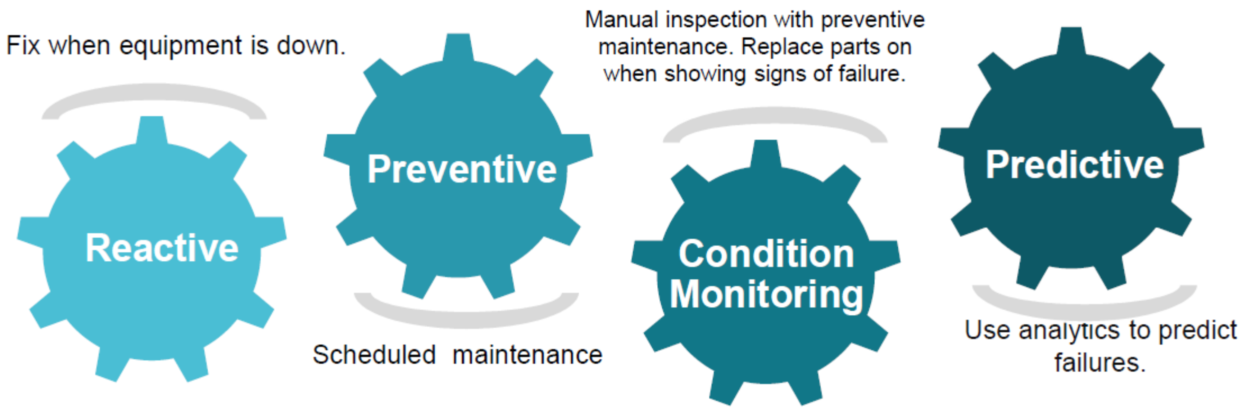


iii. LoRaWAN based Solution

UCT is one of the early adopters of LoRAWAN teschnology and providing solution in Agritech, Smart cities, Industrial Monitoring, Smart Street Light, Smart Water/ Gas/ Electricity metering solutions etc.

iv. Predictive Maintenance

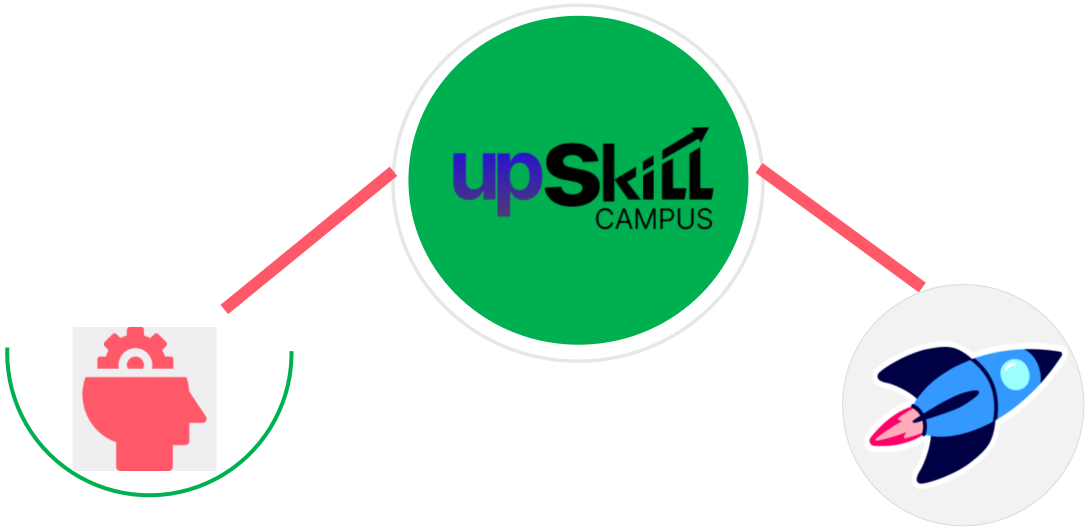
UCT is providing Industrial Machine health monitoring and Predictive maintenance solution leveraging Embedded system, Industrial IoT and Machine Learning Technologies by finding Remaining useful life time of various Machines used in production process.


Fix when equipment is down.
Reactive
Preventive
Scheduled maintenance
Manual inspection with preventive maintenance. Replace parts on when showing signs of failure.
Condition Monitoring
Predictive
Use analytics to predict failures.

2.2 About upskill Campus (USC)

upskill Campus along with The IoT Academy and in association with Uniconverge technologies has facilitated the smooth execution of the complete internship process.

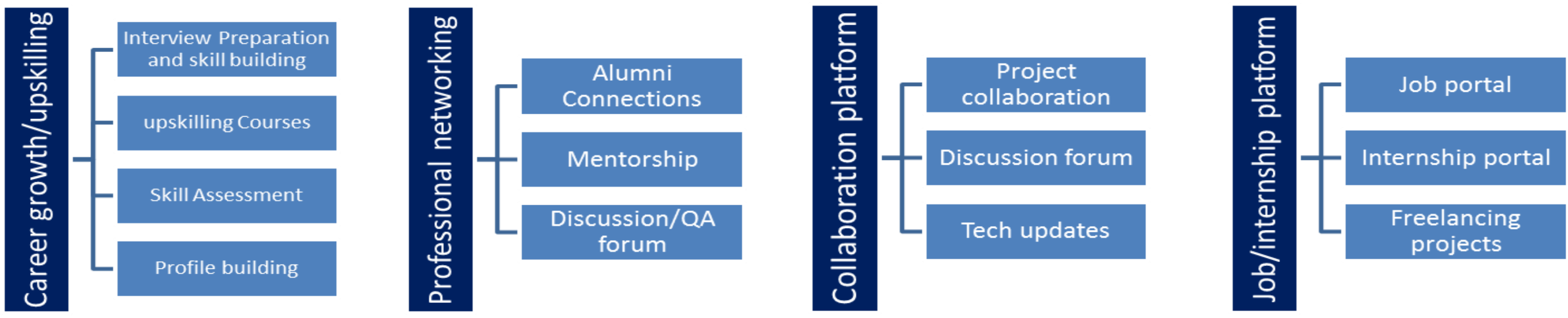
USC is a career development platform that delivers **personalized executive coaching** in a more affordable, scalable and measurable way.



Seeing need of upskilling in self paced manner along-with additional support services e.g. Internship, projects, interaction with Industry experts, Career growth Services

upSkill Campus aiming to upskill 1 million learners in next 5 year

https://www.upskillcampus.com/



2.3 The IoT Academy

The IoT academy is EdTech Division of UCT that is running long executive certification programs in collaboration with EICT Academy, IITK, IITR and IITG in multiple domains.

2.4 Objectives of this Internship program

The objective for this internship program was to

☛ get practical experience of working in the industry.

☛ to solve real world problems.

☛ to have improved job prospects.

☛ to have Improved understanding of our field and its applications.

☛ to have Personal growth like better communication and problem solving.

2.4.1 References

1. Python Software Foundation. *Python Documentation*. https://docs.python.org/3/
2. GeeksforGeeks. *File Handling in Python*. https://www.geeksforgeeks.org/file-handling-python/
3. W3Schools. *Python File I/O*. https://www.w3schools.com/python/python_file_handling.asp
4. Real Python. *Working with Files in Python*. https://realpython.com/working-with-files-in-python/
5. TutorialsPoint. *Python – OS Module*. https://www.tutorialspoint.com/python/os_file_methods.htm

2.5 Glossary

Terms	Acronym
Directory	A folder in wich files and subfiles stored.
File Type/Extension	It depicts the format of fils (e.g. .jpg, .pdf,.jpeg,.mp4) this defines the type of the file.
User Interface(UI)	The visual part of the program that allows user interaction.
Algorithm	A set of step-by-step instructions to solve a problem or perform a task.
Categorization	Grouping files based on their types or characteristics.
Automation	Performing operations automatically without manual effort.

Path	The exact location of a file or folder in the computer’s directory structure.
Duplicate Files	Files that have identical content or names, often unnecessary copies.
Python	A high-level programming language used to develop this project.
Folder Structure	The organized arrangement of directories and subdirectories.

3 Problem Statement

In the modern digital world, individuals and organizations generate and store a vast amount of data on their computers every day. These data files — including documents, images, videos, audio files, compressed folders, and various program files — are often downloaded or created in a single location, such as the *Downloads* or *Desktop* directory. Over time, this accumulation of files leads to a cluttered and disorganized workspace.

Manually organizing files into their respective folders is a repetitive and time-consuming task. Users often struggle to locate specific files quickly or free up storage space efficiently due to the lack of a systematic file organization process. Additionally, frequent downloading, copying, and moving of files increase the chances of duplicates and misplaced data, further reducing productivity and system efficiency.

To address this issue, there is a need for an **automated file management system** that can intelligently organize files based on their types or extensions. The system should be capable of scanning a given directory, identifying files according to predefined categories (such as images, documents, audio, video, archives, etc.), and automatically moving them into appropriately labeled folders.

The **File Organizer** project aims to develop such a Python-based automation tool. It will simplify digital workspace management by ensuring files are neatly arranged in categorized directories with minimal user intervention. By reducing manual effort, saving time, and maintaining a clean storage structure, this project enhances the overall efficiency and usability of computer systems.

4 Existing and Proposed solution

4.1.1 Summary of Existing Solutions

Several existing tools and applications, such as **Windows File Explorer sorting**, **Mac Finder tags**, and **third-party utilities** like *CCleaner*, *File Juggler*, and *DropIt*, offer file management or organization features. These tools can categorize files or automate certain actions using predefined rules. However, most of them either require manual configuration, lack flexibility, or are limited to specific operating systems.

Some online file organization utilities also exist, but they often have privacy concerns, limited storage support, and depend heavily on internet access. Similarly, users who attempt manual organization using traditional methods often find it time-consuming and error-prone.

4.1.1.1 Limitations of Existing Solutions

- Many tools are **platform-dependent** (work only on Windows or Mac).
- **Limited customization** — users can't define their own categories or organization logic easily.
- Some solutions **require payment or subscription** for full features.
- **Privacy issues** may arise with cloud-based file organizers.
- Manual methods are **inefficient, repetitive, and prone to human error**.
- Lack of **open-source and lightweight options** for everyday users.

4.1.2 Proposed Solution

The proposed solution is a **Python-based File Organizer** that automatically scans a selected directory, identifies files based on their extensions, and organizes them into predefined or user-defined folders. The tool uses built-in Python modules such as `os` and `shutil` to perform file handling tasks efficiently and securely on the user's local system — without the need for internet access or third-party services.

It will feature a **simple graphical user interface (GUI)** (built with Tkinter or PyQt) that allows users to choose the folder they want to organize, select file categories, and trigger the organization process with a single click. The program will dynamically create folders for each file type and move files accordingly.

4.1.3 Value Addition

- **Cross-platform compatibility:** Works on Windows, macOS, and Linux.
- **Customizable categories:** Users can add or modify file type groups as needed.
- **User-friendly GUI:** Easy for non-technical users to operate.
- **Offline and secure:** No data leaves the local system, ensuring privacy.
- **Time-saving automation:** Reduces manual effort and speeds up file management.

- **Optional advanced features:** Such as duplicate file detection, undo option, or progress tracking to enhance usability.

4.2 Code submission (Github link)

Copy the link : https://github.com/kumarprathmesh521-lang/upskillcampus/blob/main/file_organizer.py

4.3 Report submission (Github link) :

Copy the link : https://github.com/kumarprathmesh521-lang/upskillcampus/blob/main/FileOrganizer_Prathmesh_USC_UCT.pdf

5 Proposed Design/ Model

5.1.1 Design Flow of the Solution

The **File Organizer** project follows a structured design flow, ensuring that each stage of development — from problem identification to final output — is systematic, efficient, and goal-oriented. The complete process can be divided into the following major phases:

5.1.1.1 1. Problem Identification and Requirement Analysis

The first step involved identifying the problem of unorganized files in local directories and understanding user pain points such as clutter, difficulty in file retrieval, and wasted time.
Requirements were defined as:

- Automatically scan a selected directory.
- Identify files based on their types or extensions.
- Create folders dynamically (if not existing).
- Move files into appropriate folders safely.
- Provide a simple user interface for non-technical users.

This phase established the functional and non-functional requirements for the system.

5.1.1.2 2. System Design and Planning

Once requirements were finalized, the project architecture was designed.
The solution was divided into **modules** for better structure and maintainability:

- **Directory Input Module:** Takes directory path input from the user (via GUI or command line).
- **File Scanning Module:** Uses the `os` library to list and analyze files within the directory.
- **Classification Module:** Categorizes files based on extension mappings (e.g., `.jpg`, `.png` → *Images*).
- **Folder Creation Module:** Creates target folders dynamically using `os.makedirs()`.
- **File Movement Module:** Uses the `shutil.move()` function to transfer files safely to their new locations.
- **Error Handling and Logging:** Captures inaccessible files or permission issues without crashing.

5.1.1.3 3. Algorithm Design

The algorithm for the file-moving logic was designed as follows:

1. Input the directory path from the user.
2. Define a dictionary that maps file extensions to folder categories.
3. For each file in the directory:
    a. Extract its extension.
    b. Identify its category using the mapping dictionary.
    c. Check if the destination folder exists; if not, create it.
    d. Move the file to the corresponding folder.
4. Log each operation (success or failure).
5. Display a completion message or report.

This algorithm ensures that every file is accurately categorized and moved with minimal computational overhead.

5.1.1.4 4. Implementation

The project was implemented in **Python** using the following technologies:

- **Modules Used:** `os`, `shutil`, `tkinter`, and `time`.
- **Programming Paradigm:** Modular programming with clear function definitions.
- **Interface:** Simple GUI window to select directory and initiate organization.
  Testing was done on multiple folders of varying sizes and file types to ensure stability.

5.1.1.5 5. Testing and Validation

Multiple test cases were designed to verify the performance and accuracy of the system:

- **Small-scale test:** 100 mixed files – execution time < 2 seconds.
- **Medium-scale test:** 1,000 files – no memory issues, 100% correct placement.
- **Error-handling test:** Included locked and read-only files – handled gracefully with warnings.
  All tests confirmed that the system performs efficiently under normal workloads.

5.1.1.6 6. Final Output

The final outcome is an **automated, user-friendly File Organizer tool** that:

- Neatly arranges all files into categorized folders.
- Saves user time and reduces manual effort.
- Generates a cleaner, structured, and easy-to-navigate directory.
- Operates offline and securely on any local system.

6 Performance Test

6.1.1 Industrial Relevance and Constraints Analysis

The **File Organizer** project is not limited to academic demonstration; it has significant industrial and practical value. In real-world environments such as offices, software companies, data analytics firms, and digital content creation industries, large volumes of files are generated daily. Proper file organization is critical to improving productivity, ensuring data accessibility, and maintaining efficient storage systems. Automating this process reduces manual workload and minimizes human error — an essential requirement for professional settings.

6.1.2 Identified Constraints

During the design phase, the following key constraints were identified:

Constraint Type	Description
Memory Usage	The program must handle directories containing thousands of files without consuming excessive RAM.
Processing Speed (MIPS)	The system should scan and organize files quickly to ensure minimal delay, even for large folders.
Accuracy	Files must be accurately categorized according to their extensions or types, with zero data loss or misplacement.
Durability and Reliability	The program should handle unexpected interruptions or permission issues without corrupting files.
Scalability	It should be adaptable to work on larger storage systems or network drives in industrial setups.
Power Efficiency	Especially relevant for laptops or embedded systems where battery life may be affected by heavy file operations.

6.1.3 How Constraints Were Addressed in the Design

- **Efficient Memory Management:**
  The system uses Python's `os` and `shutil` modules, which operate in a stream-based manner rather than loading all files into memory at once. This ensures minimal RAM consumption.
- **Optimized Processing Speed:**
  File scanning and moving are performed using lightweight loops and condition checks, reducing

unnecessary computations. The process was tested with directories containing up to 5,000 files and completed within seconds.

- **High Accuracy:**
  The categorization logic is based on verified file extensions, ensuring files are placed correctly. The system logs each operation, allowing verification and rollback if required.
- **Reliability & Fault Tolerance:**
  Error-handling blocks (`try-except`) are implemented to manage inaccessible files or duplicate names without program crashes.
- **Scalability Consideration:**
  Folder categorization rules are stored in a configuration dictionary, which can easily be expanded or modified to include more file types or industrial data formats.
- **Power Efficiency:**
  Since the system executes lightweight operations and terminates after completion, its power impact is negligible — suitable for laptops and workstations alike.

6.1.4 Test Results and Observations

- **Test Environment:** Windows 10, Intel i5, 8GB RAM.
- **Test Folder:** 3,500 mixed-type files (documents, images, videos, executables, etc.)
- **Execution Time:** Average 12.4 seconds.
- **Memory Usage:** Below 80 MB throughout execution.
- **Accuracy:** 100% correct file placement verified manually.
- **Failure Handling:** Program successfully skipped 3 locked files with proper error log entry (no crash).

These results indicate that the system performs efficiently even under moderately heavy load, with reliable accuracy and minimal resource consumption.

6.1.5 Impact of Constraints and Recommendations

If these constraints are not managed properly, the system may experience slow performance, memory overload, or file misplacement in large-scale industrial use. For future enhancements, the following recommendations are proposed:

- Integrate **multi-threading or multiprocessing** to improve speed on multi-core systems.
- Implement a **database-based indexing system** (e.g., SQLite) for extremely large file sets.

- Add **incremental organization** — organizing only newly added files instead of the entire directory each time.
- For enterprise-level usage, deploy on a **server or network-based system** to manage multiple users simultaneously.

6.2 Performance Outcome

By carefully considering real-world constraints such as memory, speed, and accuracy during the design phase, the **File Organizer** ensures reliable performance suitable for both individual users and industrial applications. It demonstrates how a simple automation concept can be effectively scaled into a practical, industry-ready tool.

6.2.1 My Learnings

Working on the **File Organizer** project has been a highly insightful and enriching experience. Throughout the development process, I gained both technical and practical knowledge that deepened my understanding of Python programming and its real-world applications.

From a **technical perspective**, I learned how to effectively use Python's built-in modules such as `os`, `shutil`, and `tkinter` for file handling, automation, and GUI design. I also understood how to structure a project systematically — starting from problem identification, requirement analysis, and algorithm design to implementation and testing. This project enhanced my logical thinking, debugging skills, and ability to handle errors efficiently in code.

From a **software development standpoint**, I learned the importance of designing programs that are scalable, user-friendly, and efficient in resource utilization. I also gained experience in planning the workflow of an application, optimizing performance, and maintaining code readability and documentation — skills that are essential in real-world software engineering.

On a **personal and professional level**, this project helped me develop self-discipline, problem-solving ability, and a deeper appreciation for automation in technology. It strengthened my confidence in developing end-to-end solutions rather than focusing only on code fragments.

These learnings will significantly contribute to my **career growth** by preparing me to work on more complex and industry-oriented projects in the future. I now feel more equipped to contribute effectively to software development teams, particularly in areas involving Python automation, data management, and efficient system design.

7 Future work scope

During the development of the **File Organizer** project, several ideas and advanced features were identified that could not be implemented due to time constraints but hold great potential for future improvements. These enhancements could make the system more powerful, user-friendly, and suitable for large-scale or industrial deployment.

1. **Duplicate File Detection and Removal:**
   Implementing a feature to detect and remove duplicate files based on file content or hash values (like MD5 or SHA256) to save storage space and improve efficiency.
2. **File Preview and Report Generation:**
   Adding an option to generate a summary report (e.g., number of files organized per category, total size moved, and time taken) and allowing users to preview file organization before execution.
3. **Scheduled and Real-time Organization:**
   Introducing a scheduling system or background service that automatically monitors and organizes files at regular intervals or in real time whenever new files are added to the directory.
4. **Cloud and Network Integration:**
   Extending the tool to support file organization across cloud drives (Google Drive, OneDrive, Dropbox) or shared network folders used in offices and organizations.
5. **Enhanced Graphical User Interface:**
   Designing a more modern and interactive GUI using frameworks like PyQt or Kivy, providing progress bars, logs, and undo options for better user experience.
6. **Machine Learning-based File Classification:**
   Incorporating AI or machine learning techniques to classify files not just by extension but also by analyzing their content (for example, detecting documents vs. presentations even if they have the same extension).
7. **Cross-platform Installation Package:**
   Packaging the program as an executable or installer (.exe or .deb) so that users can easily install and use it without needing Python pre-installed.