Bahria University Islamabad Campus

Department of Computer Science

Class/Section: BS (AI)-5A

PROJECT REPORT

Class: MACHINE LEARNING **Date**: 03/06/2024

NAME	ENROLLMENT
FARHAN AHMAD	01-136221-052
TAHA HASNAT	01-136221-018

COURSE:

MACHINE LEARNING LAB

SUBMITTED TO:

DR. ASGHAR ALI SHAH

TOPIC:

RESUME SCREENER

INTRODUCTION

In today's competitive job market, recruiters are often overwhelmed by the sheer volume of resumes they receive for each job opening, making manual screening a daunting and error-prone task. To streamline this process, we have developed an advanced Resume Screener that utilizes machine learning to automate the categorization of resumes. This system extracts text from PDF resumes, cleans and processes the content, and employs a machine learning pipeline to classify resumes into specific job categories. By automating the initial screening phase, our Resume Screener significantly reduces the workload on human resources personnel, enhances the efficiency of the recruitment process, and ensures a more accurate and unbiased evaluation of candidates.

PROJECT SCOPE

The scope of this Resume Screener project encompasses the following key areas and functionalities:

1. Text Extraction from PDF Resumes: Implementing a robust mechanism using the PyPDF2 library to accurately extract text from resumes in PDF format.

2. Text Cleaning and Preprocessing: Developing comprehensive text cleaning functions to remove unnecessary elements such as headers, footers, special characters, and irrelevant patterns. Additionally, extracting and processing relevant sections of the resume, such as qualifications and experience, to form a coherent and useful input for the machine learning model.

3. Machine Learning Model Development: Constructing a machine learning pipeline that includes a TF-IDF Vectorizer for text vectorization and a

RandomForestClassifier for categorizing resumes into predefined job categories. The model is trained using a labeled dataset of resumes to ensure accurate and reliable predictions.

4. Handling Missing Data: Implementing strategies to manage and impute missing values within the resume data to maintain the integrity and quality of the input for the machine learning model.

5. Graphical User Interface (GUI): Designing and developing a user-friendly interface using Tkinter to allow users to easily upload folders containing

   resumes, initiate the screening process, and view categorized outputs. The GUI provides clear instructions and feedback to ensure a smooth user experience.

6. File Management and Organization: Automating the organization of categorized resumes by creating directories for each job category and moving the processed resumes into their respective folders. This feature simplifies the management and retrieval of categorized resumes.

7. Logging and Error Handling: Implementing logging mechanisms to record the system's operations and error handling procedures to manage exceptions, ensuring the robustness and reliability of the application.

8. Documentation and User Instructions: Providing comprehensive documentation and user instructions to guide users on how to install, configure, and use the Resume Screener effectively.

By covering these areas, the Resume Screener project aims to deliver a complete and efficient solution for automating the initial phase of resume screening, thereby enhancing the overall efficiency and accuracy of the recruitment process.

PROJECT OBJECTIVES

The Resume Screener project aims to achieve the following objectives:

1. Automate Resume Screening:

Develop a system that can automatically extract and process text from PDF resumes, reducing the need for manual screening.

2. <u>Enhance Screening Efficiency:</u>

   Implement a machine learning pipeline to quickly and accurately categorize resumes into predefined job categories, significantly speeding up the initial phase of recruitment.

3. <u>Improve Screening Accuracy:</u>

   Utilize advanced text preprocessing and machine learning techniques to ensure high accuracy in resume classification, thereby minimizing errors and biases in the screening process.

4. <u>User-Friendly Interface:</u>

   Design a graphical user interface (GUI) that allows recruiters to easily upload and process folders of resumes, making the system accessible even to non-technical users.

5. <u>Effective Data Handling:</u>

   Implement robust mechanisms for handling missing data and ensuring the integrity and quality of input data used for training and predictions.

6. <u>Seamless File Organization:</u>

Automate the organization of categorized resumes into designated folders, simplifying the management and retrieval of screened resumes.

7. Reliability and Robustness:

Incorporate comprehensive logging and error handling to ensure the system operates reliably and can effectively manage any issues that arise during processing.

8. Provide Comprehensive Documentation:

Develop detailed documentation and user instructions to facilitate the installation, configuration, and use of the Resume Screener, ensuring users can fully leverage its capabilities.

9. Scalability and Extensibility:

Design the system with scalability and extensibility in mind, allowing for future enhancements and the ability to handle increasing volumes of resumes as needed.

10. Support HR Efficiency:

Ultimately, reduce the workload on human resources personnel by providing an automated solution that enhances the efficiency and accuracy of the recruitment process, allowing HR teams to focus on more strategic tasks.

By achieving these objectives, the Resume Screener project will deliver a powerful tool to streamline and improve the resume screening process, benefiting both recruiters and job applicants.

MAIN FETURES & FUNCTIONALITIES

The Resume Screener project incorporates a range of features and functionalities designed to streamline and enhance the resume screening process. The key features and functionalities are outlined below:

PDF TEXT EXTRACTION

Feature: Ability to extract text from resumes in PDF format using the PyPDF2 library.
Functionality: The system processes PDF files to extract textual content from each page, ensuring that all relevant information is captured for further analysis.

TEXT CLEANING AND PREPROCESSING

Feature: Comprehensive text cleaning to remove irrelevant elements such as headers, footers, and special characters.

Functionality: The system cleans the extracted text to ensure that it is in a consistent and useful format for machine learning. It also extracts and processes specific sections like qualifications and experience.

MACHINE LEARNING CLASSIFICATION:

Feature: Advanced resume categorization using a machine learning pipeline.

Functionality: The pipeline includes a TF-IDF Vectorizer for text vectorization and a RandomForestClassifier for categorizing resumes into predefined job categories. The model is trained on a labeled dataset to ensure high accuracy in predictions.

HANDLING MISSING DATA:

Feature: Strategies to manage and impute missing data.

Functionality: The system uses imputation techniques to handle any missing values in the resume data, maintaining data integrity and quality for accurate predictions.

GRAPHICAL USER INTERFACE (GUI)

Feature: User-friendly interface for uploading and processing resumes.

Functionality: The GUI, built using Tkinter, allows users to select and upload folders containing resumes, initiate the screening process, and view categorized outputs. It provides clear instructions and feedback to ensure ease of use.

AUTOMATED FILE ORGANIZATION

Feature: Automatic categorization and organization of resumes.

Functionality: After categorizing resumes, the system creates directories for each job category and moves the processed resumes into their respective folders. This automation simplifies file management and retrieval.

LOGGING AND ERROR HANDLING

Feature: Comprehensive logging and error handling mechanisms.

Functionality: The system logs all operations and includes robust error handling to manage exceptions. This ensures reliability and helps in diagnosing and resolving issues promptly.

SCALABILITY AND EXTENSIBILITY

Feature: Design for scalability and future enhancements.
Functionality: The system is built to handle increasing volumes of resumes and can be extended with additional features or improved models as needed.

DOCUMENTATION AND USER INSTRUCTIONS:

Feature: Detailed documentation and user guides.
Functionality: Comprehensive documentation is provided to assist users in installing, configuring, and using the Resume Screener effectively. This includes step-by-step instructions and troubleshooting tips.

SUPPORT FOR HR EFFICIENCY

Feature: Enhancements to HR efficiency through automation.
Functionality: By automating the initial screening of resumes, the system reduces the workload on HR personnel, allowing them to focus on more strategic tasks and improving the overall efficiency of the recruitment process.

These features and functionalities ensure that the Resume Screener project provides a complete, efficient, and user-friendly solution for automating the resume screening process, thereby enhancing the accuracy and efficiency of recruitment efforts.

FLOW OF PROGRAM

1. INITIALIZATION
   Load required libraries
   Setup logging

2. PRE-TRAINED MODEL LOADING
   Load the pre-trained machine learning pipeline model (`pipeline.joblib`)

3. <u>GRAPHICAL USER INTERFACE (GUI) SETUP</u>
   Initialize GUI with `tkinter`
   Provide options for folder selection and uploading
   Setup text widget for displaying categorized results

4. <u>USER INTERACTION</u>
   Select folder containing PDF files
   Enable "Upload Folder" button upon selection

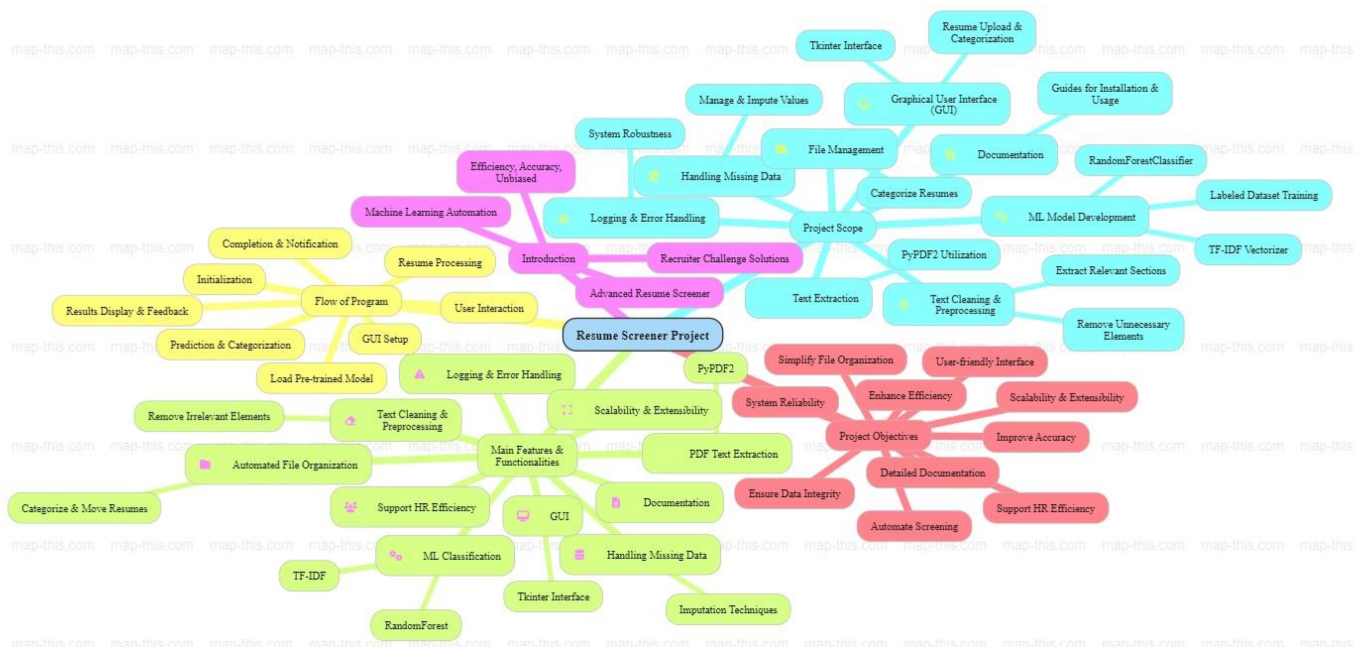
5. <u>RESUME PROCESSING</u>
   Read and extract text from PDF files using `PyPDF2`
   Clean text to remove unwanted characters and standardize format
   Preprocess text by extracting specific sections and concatenating text
   Handle missing values with imputation

6. <u>PREDICTION AND CATEGORIZATION</u>
   Predict job category using the loaded machine learning pipeline
   Create directories for each predicted category
   Move processed PDFs to respective category folders

7. <u>RESULTS DISPLAY AND FEEDBACK</u>
   Display categorization results in the GUI
   Show summary of categorized folders and number of resumes in each category

8. <u>COMPLETION AND USER NOTIFICATION</u>

Re-enable "Upload Folder" button after processing
Display success message indicating documents have been categorized successfully.

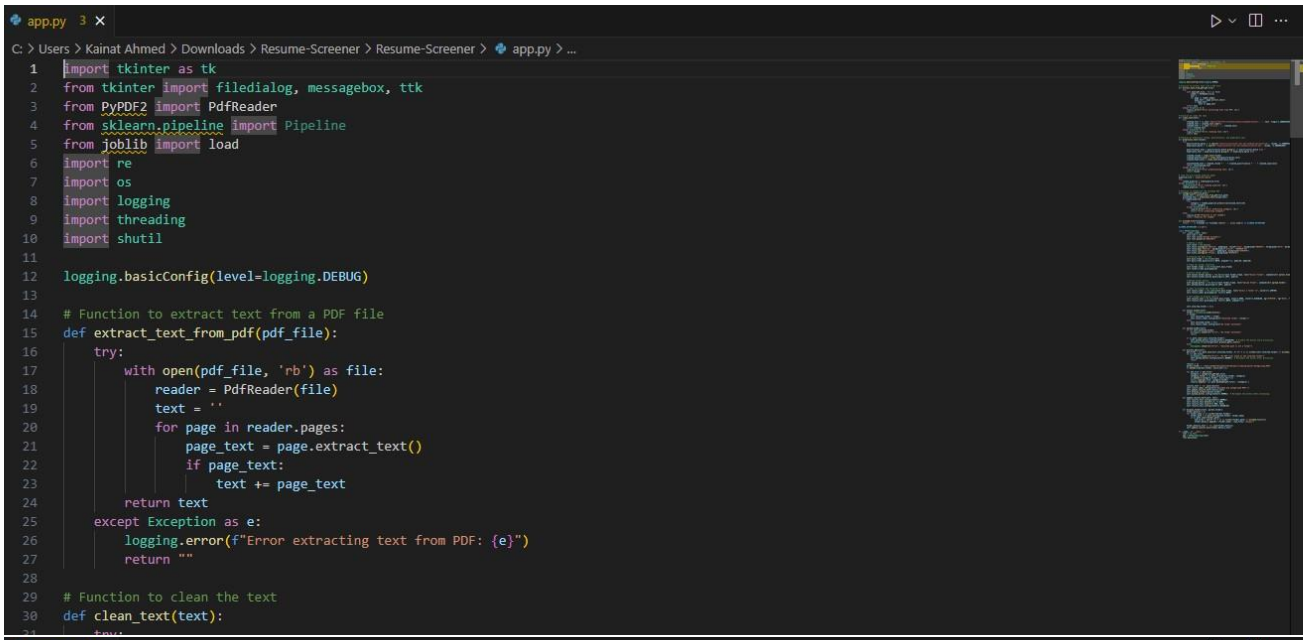


CODE IMPLIMENTATIONS

APP. PY

```
import tkinter as tk
from tkinter import filedialog, messagebox, ttk
from PyPDF2 import PdfReader
from sklearn.pipeline import Pipeline
from joblib import load
import re
import os
import logging
import threading
import shutil

logging.basicConfig(level=logging.DEBUG)

# Function to extract text from a PDF file
def extract_text_from_pdf(pdf_file):
    try:
        with open(pdf_file, 'rb') as file:
            reader = PdfReader(file)
            text = ''
            for page in reader.pages:
                page_text = page.extract_text()
                if page_text:
                    text += page_text
        return text
    except Exception as e:
        logging.error(f"Error extracting text from PDF: {e}")
        return ""

# Function to clean the text
def clean_text(text):
    try:
        cleaned_text = re.sub(r'Page\s+\d+\s+of\s+\d+|Confidential|Header|Footer', '', text,
flags=re.IGNORECASE)
        cleaned_text = cleaned_text.lower()
        cleaned_text = re.sub(r'[^\w\s]', '', cleaned_text)
        return cleaned_text
    except Exception as e:
        logging.error(f"Error cleaning text: {e}")
        return text

# Function to preprocess resume, qualification, and experience text
def preprocess_text(resume):
    try:
        qualification_match = re.search(r'Qualification[s]?:\s*(.*?)\s(Experience[s]?:|$)', resume,
re.IGNORECASE)
        experience_match = re.search(r'Experience[s]?:\s*(.*?)\s(Education[s]?:|$)', resume,
re.IGNORECASE)

        qualification_text = qualification_match.group(1) if qualification_match else ''
        experience_text = experience_match.group(1) if experience_match else ''

        cleaned_resume = clean_text(resume)
        cleaned_qualification = clean_text(qualification_text)
        cleaned_experience = clean_text(experience_text)

        concatenated_text = cleaned_resume + ' ' + cleaned_qualification + ' ' + cleaned_experience
        return concatenated_text
    except Exception as e:
        logging.error(f"Error preprocessing text: {e}")
        return resume

# Load the pre-trained pipeline model
pipeline_file = 'pipeline.joblib'
loaded_pipeline = None
```

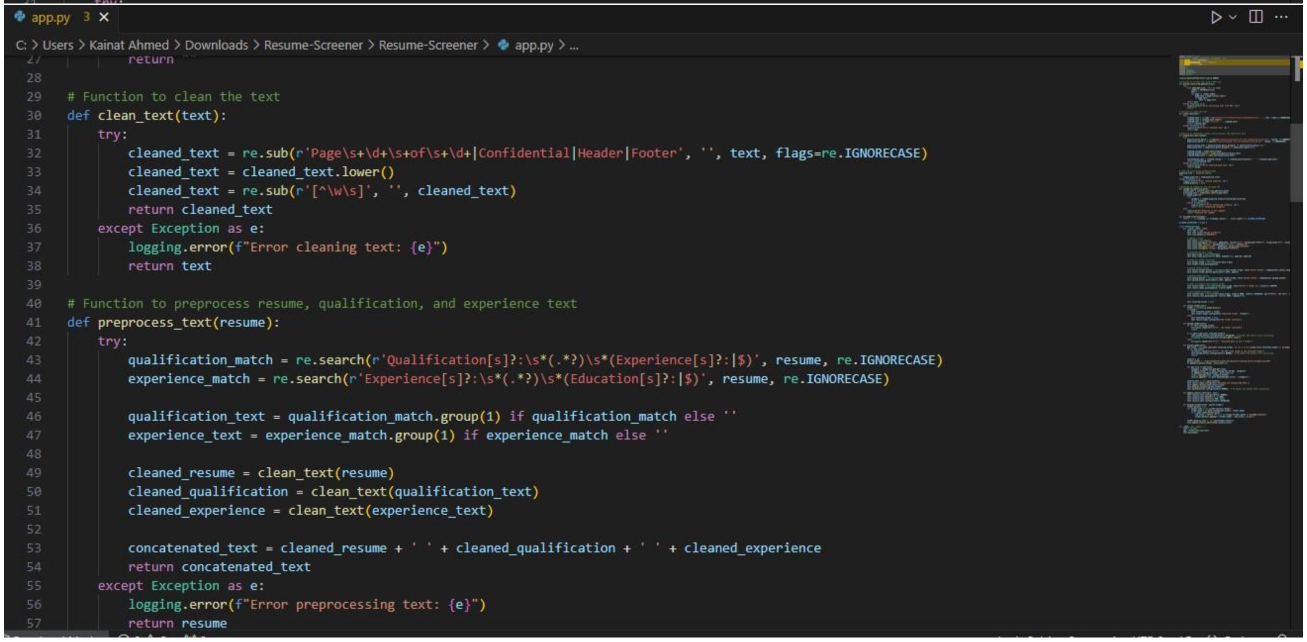
```
try:
    loaded_pipeline = load(pipeline_file)
    logging.info("Pipeline loaded successfully")
except Exception as e:
    logging.error(f"Error loading pipeline: {e}")

# Function to categorize the uploaded PDF
def categorize_pdf(file_path):
    resume_text = extract_text_from_pdf(file_path)
    processed_text = preprocess_text(resume_text)
    if loaded_pipeline:
        try:
            category = loaded_pipeline.predict([processed_text])[0]
            return category
        except Exception as e:
            logging.error(f"Error predicting category: {e}")
            return "Error predicting category"
    else:
        logging.error("Pipeline is not loaded")
        return "Pipeline not loaded"

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

ALLOWED_EXTENSIONS = {'pdf'}

class PDFUploaderApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Resume Screener")
        self.root.geometry("600x400")

        # Adding a style
        self.style = ttk.Style()
        self.style.configure("TButton", padding=6, relief="flat", background="#007bff",
foreground="#000000", bordercolor="#007bff")
        self.style.map("TButton", background=[("active", "#0069d9")])
        self.style.configure("TLabel", padding=6, background="#f9f9f9")
        self.style.configure("TFrame", background="#f9f9f9")

        # Creating the main frame
        self.main_frame = ttk.Frame(root)
        self.main_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)
        # Frame for folder selection
        self.folder_frame = ttk.Frame(self.main_frame)
        self.folder_frame.pack(pady=10)

        # Select folder button
        self.select_folder_button = ttk.Button(self.folder_frame, text="Select Folder",
command=self.select_folder)
        self.select_folder_button.pack(side=tk.LEFT, padx=5)

        # Upload folder button
        self.upload_button = ttk.Button(self.folder_frame, text="Upload Folder",
command=self.upload_folder)
        self.upload_button.pack(side=tk.LEFT, padx=5)

        # Label to display the selected folder
        self.result_label = ttk.Label(self.main_frame, text="Select a folder to categorize PDFs",
anchor=tk.CENTER)
        self.result_label.pack(pady=10, fill=tk.BOTH)

        # Text widget to display results
        self.results_text = tk.Text(self.main_frame, wrap=tk.WORD, state=tk.DISABLED, bg="#f0f0f0",
fg="#333", font=("Helvetica", 10))
        self.results_text.pack(pady=10, fill=tk.BOTH, expand=True)

        self.selected_folder = None

        # Label to display the creators' names
        self.creators_label = ttk.Label(self.main_frame, text="Made by FARHAN AHMAD AND TAHA
HASNAT", anchor=tk.CENTER, background="#f9f9f9", font=("Helvetica", 10, "italic"))
        self.creators_label.pack(side=tk.BOTTOM, pady=10)

    def select_folder(self):
        folder = filedialog.askdirectory()
        if folder:
            self.selected_folder = folder
            self.result_label.config(text=f"Selected folder: {folder}")
        else:
            self.selected_folder = None
            self.result_label.config(text="No folder selected")

    def upload_folder(self):
        if not self.selected_folder:
            messagebox.showerror("Error", "No folder selected")
            return
        if os.path.isdir(self.selected_folder):
            self.upload_button.config(state=tk.DISABLED)  # Disable the button while processing
            threading.Thread(target=self.process_pdfs).start()
        else:
            messagebox.showerror("Error", "Selected path is not a folder")

    def process_pdfs(self):
        pdf_files = [os.path.join(self.selected_folder, f) for f in os.listdir(self.selected_folder) if
allowed_file(f)]
        if not pdf_files:
            messagebox.showerror("Error", "No PDF files found in the selected folder")
            self.upload_button.config(state=tk.NORMAL)  # Re-enable the button after processing
            return

        results = []
        parent_folder = 'C:/Users/Kainat Ahmed/Desktop/Resume-Screener/Categorized_PDFs'
        os.makedirs(parent_folder, exist_ok=True)

        for pdf_file in pdf_files:
            category = categorize_pdf(pdf_file)
            category_folder = os.path.join(parent_folder, category)
            os.makedirs(category_folder, exist_ok=True)
            shutil.copy(pdf_file, category_folder)
            results.append(f'{os.path.basename(pdf_file)}: {category}')

        results_text = '\n'.join(results)
        self.result_label.config(text="Uploaded and Categorized PDFs:")
        self.update_results_text(results_text)
        self.display_folders(parent_folder)
        self.upload_button.config(state=tk.NORMAL)  # Re-enable the button after processing

    def update_results_text(self, text):
        self.results_text.config(state=tk.NORMAL)
        self.results_text.delete(1.0, tk.END)
        self.results_text.insert(tk.END, text)
        self.results_text.config(state=tk.DISABLED)

    def display_folders(self, parent_folder):
        folder_details = []
        for folder_name in os.listdir(parent_folder):
            folder_path = os.path.join(parent_folder, folder_name)
            if os.path.isdir(folder_path):
                num_files = len([f for f in os.listdir(folder_path) if allowed_file(f)])
            folder_details.append(f'{folder_name}: {num_files} resumes')

        folder_details_text = '\n'.join(folder_details)
        self.update_results_text(folder_details_text)

if __name__ == '__main__':
    root = tk.Tk()
    app = PDFUploaderApp(root)
    root.mainloop()
```



app.py 3 ×

C: > Users > Kainat Ahmed > Downloads > Resume-Screener > Resume-Screener > app.py > ...

```python
import tkinter as tk
from tkinter import filedialog, messagebox, ttk
from PyPDF2 import PdfReader
from sklearn.pipeline import Pipeline
from joblib import load
import re
import os
import logging
import threading
import shutil

logging.basicConfig(level=logging.DEBUG)

# Function to extract text from a PDF file
def extract_text_from_pdf(pdf_file):
    try:
        with open(pdf_file, 'rb') as file:
            reader = PdfReader(file)
            text = ''
            for page in reader.pages:
                page_text = page.extract_text()
                if page_text:
                    text += page_text
        return text
    except Exception as e:
        logging.error(f"Error extracting text from PDF: {e}")
        return ""

# Function to clean the text
def clean_text(text):
```

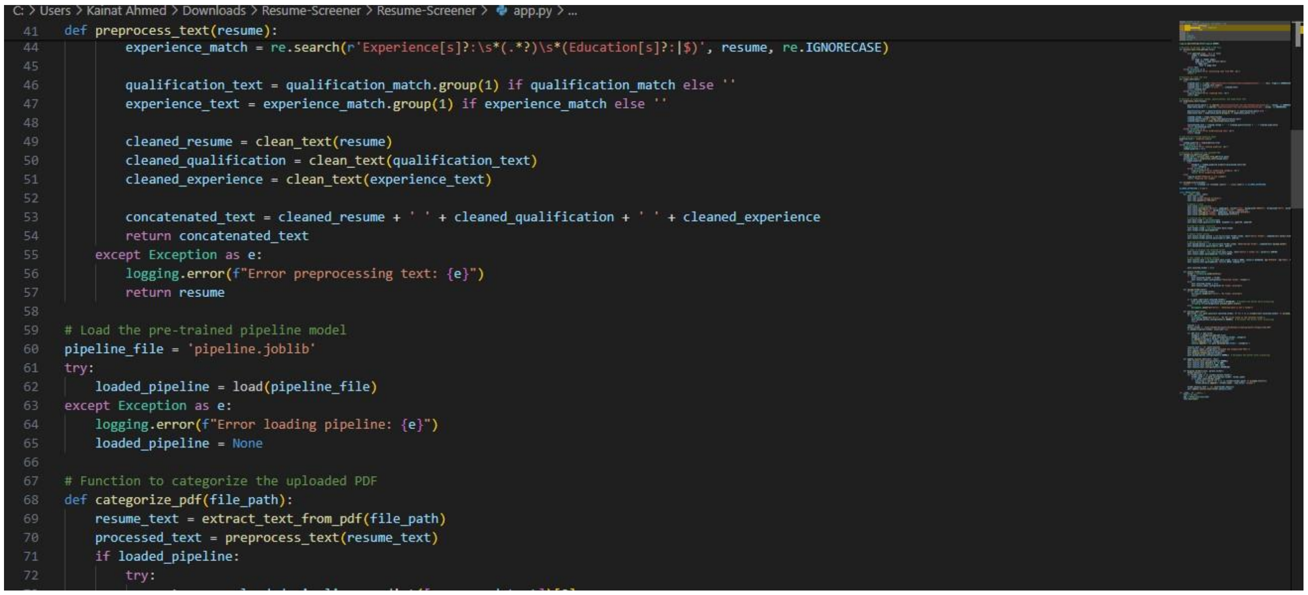


app.py 3 ×

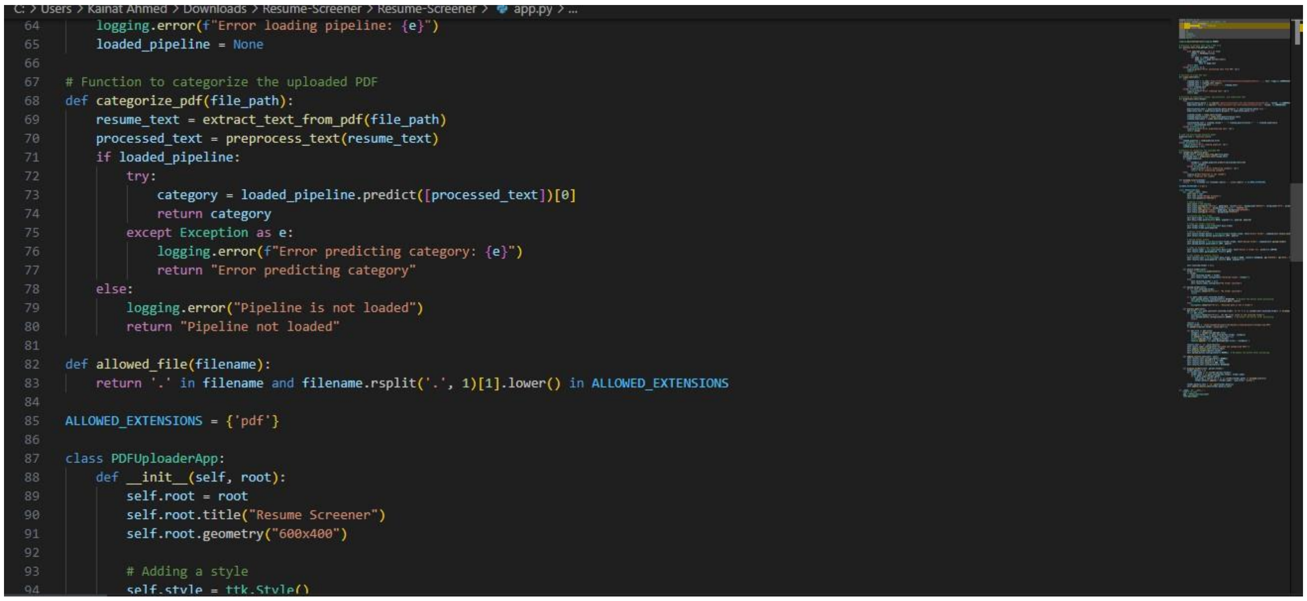
C: > Users > Kainat Ahmed > Downloads > Resume-Screener > Resume-Screener > app.py > ...

```python
# Function to clean the text
def clean_text(text):
```

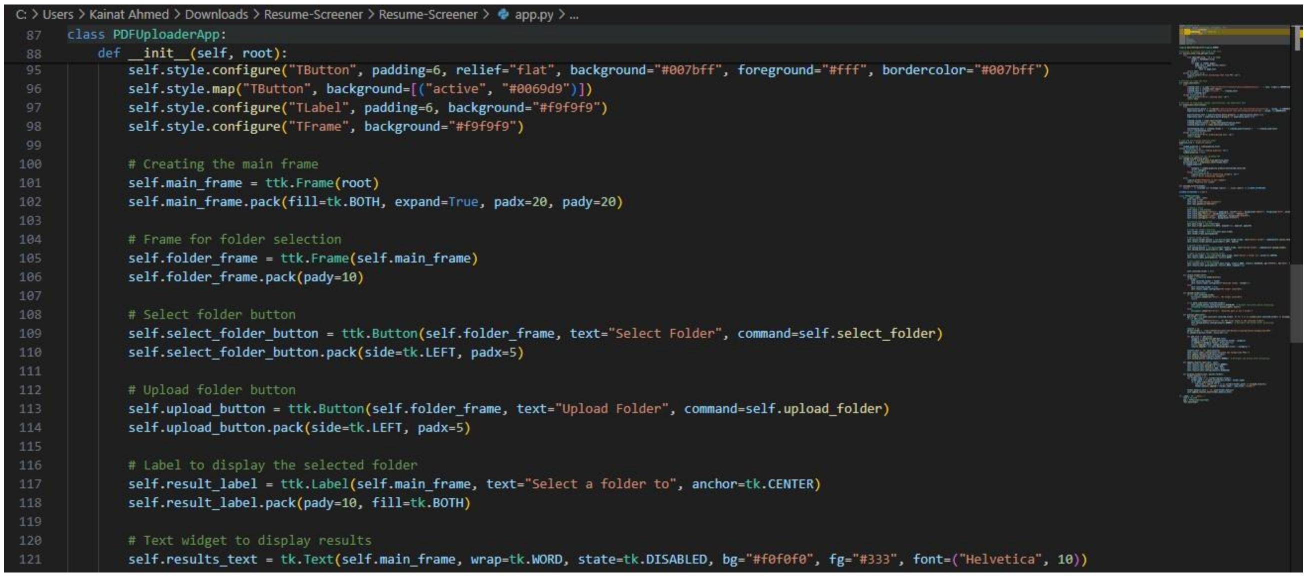
```python
    try:
        cleaned_text = re.sub(r'Page\s+\d+\s+of\s+\d+|Confidential|Header|Footer', '', text, flags=re.IGNORECASE)
        cleaned_text = cleaned_text.lower()
        cleaned_text = re.sub(r'[^\w\s]', '', cleaned_text)
        return cleaned_text
    except Exception as e:
        logging.error(f"Error cleaning text: {e}")
        return text

# Function to preprocess resume, qualification, and experience text
def preprocess_text(resume):
    try:
        qualification_match = re.search(r'Qualification[s]?:\s*(.*?)\s*(Experience[s]?:|$)', resume, re.IGNORECASE)
        experience_match = re.search(r'Experience[s]?:\s*(.*?)\s*(Education[s]?:|$)', resume, re.IGNORECASE)

        qualification_text = qualification_match.group(1) if qualification_match else ''
        experience_text = experience_match.group(1) if experience_match else ''

        cleaned_resume = clean_text(resume)
        cleaned_qualification = clean_text(qualification_text)
        cleaned_experience = clean_text(experience_text)

        concatenated_text = cleaned_resume + ' ' + cleaned_qualification + ' ' + cleaned_experience
        return concatenated_text
    except Exception as e:
        logging.error(f"Error preprocessing text: {e}")
        return resume
```



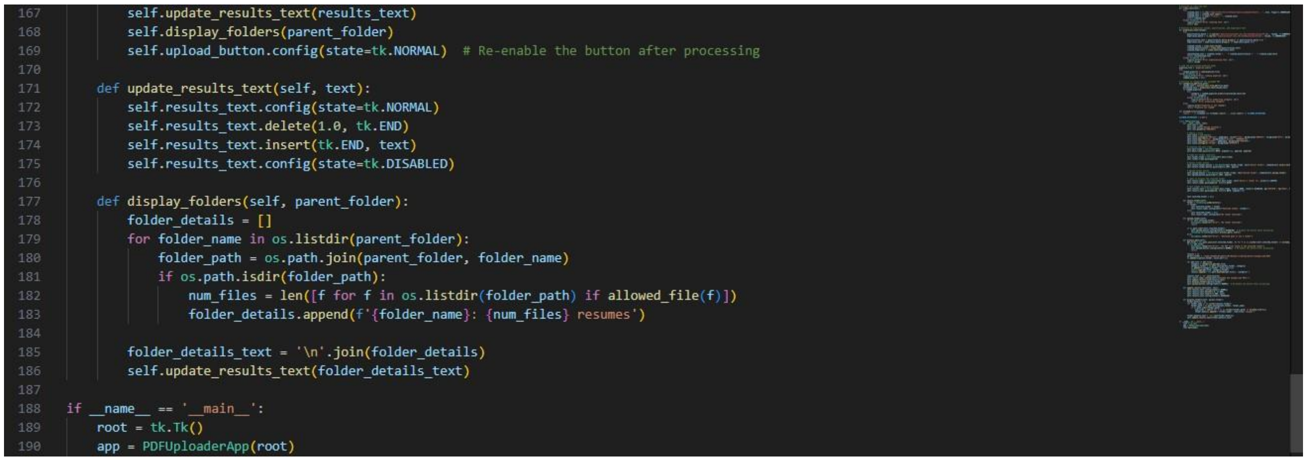
```
C: > Users > Kainat Ahmed > Downloads > Resume-Screener > Resume-Screener > app.py > ...
def preprocess_text(resume):
        experience_match = re.search(r'Experience[s]?:\s*(.*?)\s*(Education[s]?:|$)', resume, re.IGNORECASE)

        qualification_text = qualification_match.group(1) if qualification_match else ''
        experience_text = experience_match.group(1) if experience_match else ''

        cleaned_resume = clean_text(resume)
        cleaned_qualification = clean_text(qualification_text)
        cleaned_experience = clean_text(experience_text)

        concatenated_text = cleaned_resume + ' ' + cleaned_qualification + ' ' + cleaned_experience
        return concatenated_text
    except Exception as e:
        logging.error(f"Error preprocessing text: {e}")
        return resume

# Load the pre-trained pipeline model
pipeline_file = 'pipeline.joblib'
try:
    loaded_pipeline = load(pipeline_file)
except Exception as e:
    logging.error(f"Error loading pipeline: {e}")
    loaded_pipeline = None

# Function to categorize the uploaded PDF
def categorize_pdf(file_path):
    resume_text = extract_text_from_pdf(file_path)
    processed_text = preprocess_text(resume_text)
    if loaded_pipeline:
        try:
```



```
C: > Users > Kainat Ahmed > Downloads > Resume-Screener > Resume-Screener > app.py > ...
    logging.error(f"Error loading pipeline: {e}")
    loaded_pipeline = None

# Function to categorize the uploaded PDF
def categorize_pdf(file_path):
    resume_text = extract_text_from_pdf(file_path)
    processed_text = preprocess_text(resume_text)
    if loaded_pipeline:
        try:
            category = loaded_pipeline.predict([processed_text])[0]
            return category
        except Exception as e:
            logging.error(f"Error predicting category: {e}")
            return "Error predicting category"
    else:
        logging.error("Pipeline is not loaded")
        return "Pipeline not loaded"

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

ALLOWED_EXTENSIONS = {'pdf'}

class PDFUploaderApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Resume Screener")
        self.root.geometry("600x400")

        # Adding a style
        self.style = ttk.Style()
```


C: > Users > Kainat Ahmed > Downloads > Resume-Screener > Resume-Screener > app.py > ...

```python
class PDFUploaderApp:
    def __init__(self, root):
        self.style.configure("TButton", padding=6, relief="flat", background="#007bff", foreground="#fff", bordercolor="#007bff")
        self.style.map("TButton", background=[("active", "#0069d9")])
        self.style.configure("TLabel", padding=6, background="#f9f9f9")
        self.style.configure("TFrame", background="#f9f9f9")

        # Creating the main frame
        self.main_frame = ttk.Frame(root)
        self.main_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=20)

        # Frame for folder selection
        self.folder_frame = ttk.Frame(self.main_frame)
        self.folder_frame.pack(pady=10)

        # Select folder button
        self.select_folder_button = ttk.Button(self.folder_frame, text="Select Folder", command=self.select_folder)
        self.select_folder_button.pack(side=tk.LEFT, padx=5)

        # Upload folder button
        self.upload_button = ttk.Button(self.folder_frame, text="Upload Folder", command=self.upload_folder)
        self.upload_button.pack(side=tk.LEFT, padx=5)

        # Label to display the selected folder
        self.result_label = ttk.Label(self.main_frame, text="Select a folder to", anchor=tk.CENTER)
        self.result_label.pack(pady=10, fill=tk.BOTH)

        # Text widget to display results
        self.results_text = tk.Text(self.main_frame, wrap=tk.WORD, state=tk.DISABLED, bg="#f0f0f0", fg="#333", font=("Helvetica", 10))
```



```python
            self.update_results_text(results_text)
            self.display_folders(parent_folder)
            self.upload_button.config(state=tk.NORMAL)  # Re-enable the button after processing

    def update_results_text(self, text):
        self.results_text.config(state=tk.NORMAL)
        self.results_text.delete(1.0, tk.END)
        self.results_text.insert(tk.END, text)
        self.results_text.config(state=tk.DISABLED)

    def display_folders(self, parent_folder):
        folder_details = []
        for folder_name in os.listdir(parent_folder):
            folder_path = os.path.join(parent_folder, folder_name)
            if os.path.isdir(folder_path):
                num_files = len([f for f in os.listdir(folder_path) if allowed_file(f)])
                folder_details.append(f'{folder_name}: {num_files} resumes')

        folder_details_text = '\n'.join(folder_details)
        self.update_results_text(folder_details_text)

if __name__ == '__main__':
    root = tk.Tk()
    app = PDFUploaderApp(root)
```

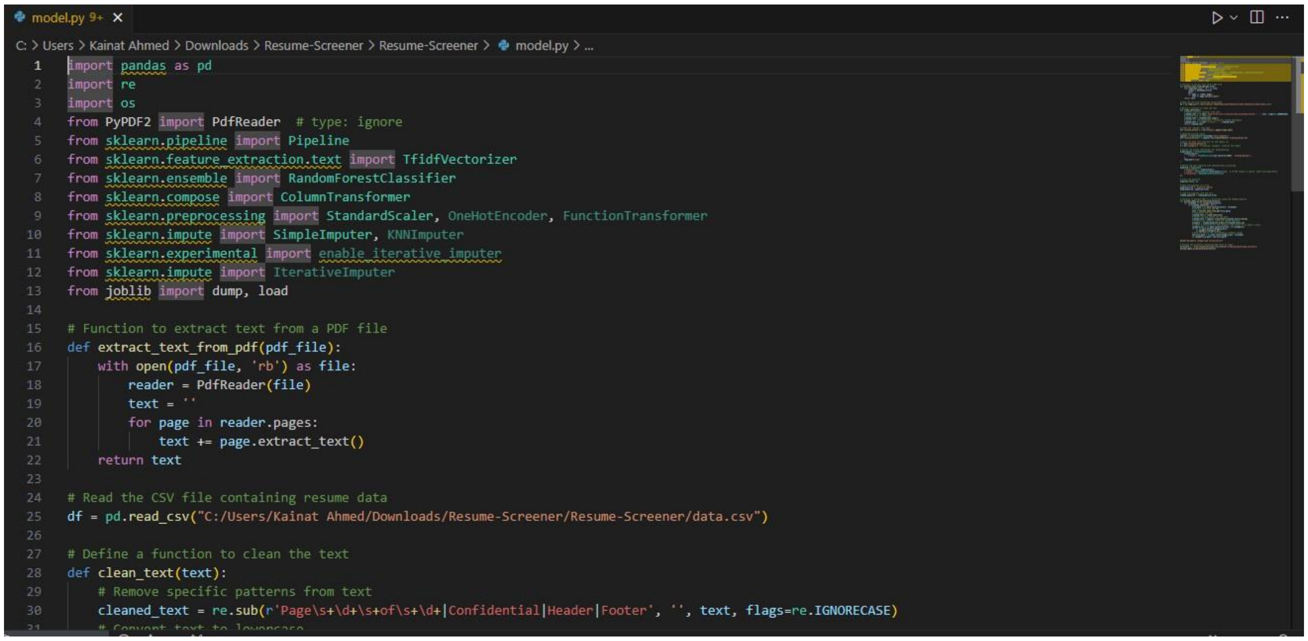
MODAL.PY

```
import pandas as pd

import re

import os

from PyPDF2 import PdfReader  # type: ignore

from sklearn.pipeline import Pipeline

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.ensemble import RandomForestClassifier

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from joblib import dump, load


# Function to extract text from a PDF file

def extract_text_from_pdf(pdf_file):

    with open(pdf_file, 'rb') as file:

        reader = PdfReader(file)

        text = ''

        for page in reader.pages:

            text += page.extract_text()

    return text


# Read the CSV file containing resume data

df = pd.read_csv("C:/Users/Kainat Ahmed/Desktop/Resume-Screener/data.csv")
```

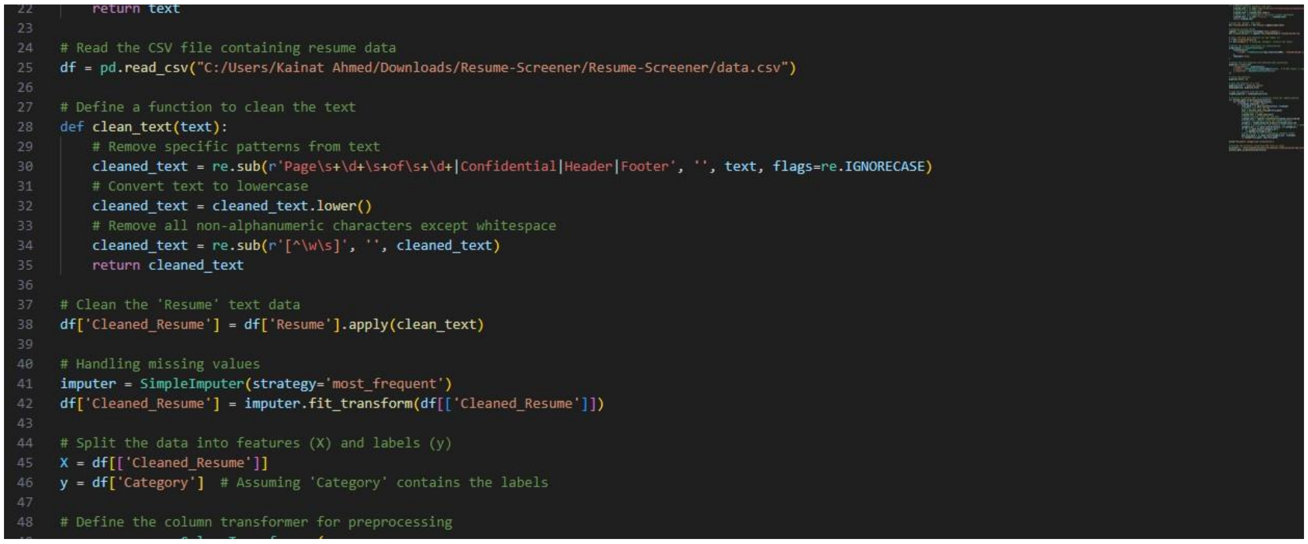
```
# Define a function to clean the text

def clean_text(text):

    # Remove specific patterns from text

    cleaned_text = re.sub(r'Page\s+\d+\s+of\s+\d+|Confidential|Header|Footer', '', text, flags=re.IGNORECASE)

    # Convert text to lowercase

    cleaned_text = cleaned_text.lower()

    # Remove all non-alphanumeric characters except whitespace

    cleaned_text = re.sub(r'[^\w\s]', '', cleaned_text)

    return cleaned_text


# Clean the 'Resume' text data

df['Cleaned_Resume'] = df['Resume'].apply(clean_text)



# Handling missing values

imputer = SimpleImputer(strategy='most_frequent')

df['Cleaned_Resume'] = imputer.fit_transform(df[['Cleaned_Resume']])[:, 0]



# Split the data into features (X) and labels (y)

X = df[['Cleaned_Resume']]

y = df['Category']  # Assuming 'Category' contains the labels



# Define the column transformer for preprocessing

preprocessor = ColumnTransformer(
    transformers=[
        ('tfidf', TfidfVectorizer(max_features=1000), 'Cleaned_Resume'),
    ],
    remainder='drop'
)

# Define the full pipeline with advanced data processing
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier())
])

# Train the pipeline
pipeline.fit(X, y)

# Save the pipeline to a file
pipeline_file = 'pipeline.joblib'
dump(pipeline, pipeline_file)

# Load the pipeline from the file
loaded_pipeline = load(pipeline_file)

# Function to process PDFs in a directory using the loaded pipeline
```

```
def process_pdfs_in_directory(directory):
    for filename in os.listdir(directory):
        if filename.endswith('.pdf'):
            file_path = os.path.join(directory, filename)
            # Extract text from the PDF
            text = extract_text_from_pdf(file_path)
            # Preprocess the text
            cleaned_text = clean_text(text)
            # Handle missing values for new data
            cleaned_text = imputer.transform([[cleaned_text]])[0][0]
            # Predict the category using the loaded pipeline
            category = loaded_pipeline.predict([cleaned_text])[0]
            # Create a directory for the category (job title) if it doesn't exist
            category_dir = os.path.join(directory, str(category))
            if not os.path.exists(category_dir):
                os.makedirs(category_dir)
            # Move the PDF to the corresponding category folder
            new_file_path = os.path.join(category_dir, filename)
            os.rename(file_path, new_file_path)

print("Documents categorized successfully")

# Provide the directory containing PDF files as input
```

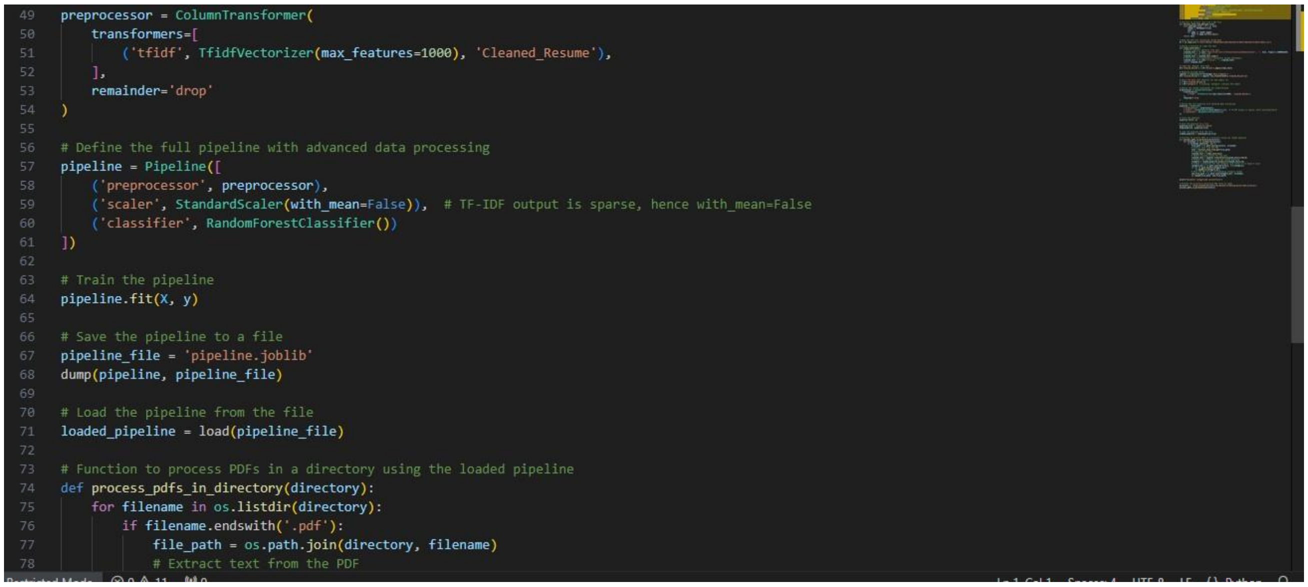
```
directory = "C:/Users/Kainat Ahmed/Desktop/Resume-Screener/Categorized_PDFs"

process_pdfs_in_directory(directory)
```


model.py 9+
C: > Users > Kainat Ahmed > Downloads > Resume-Screener > Resume-Screener > model.py > ...

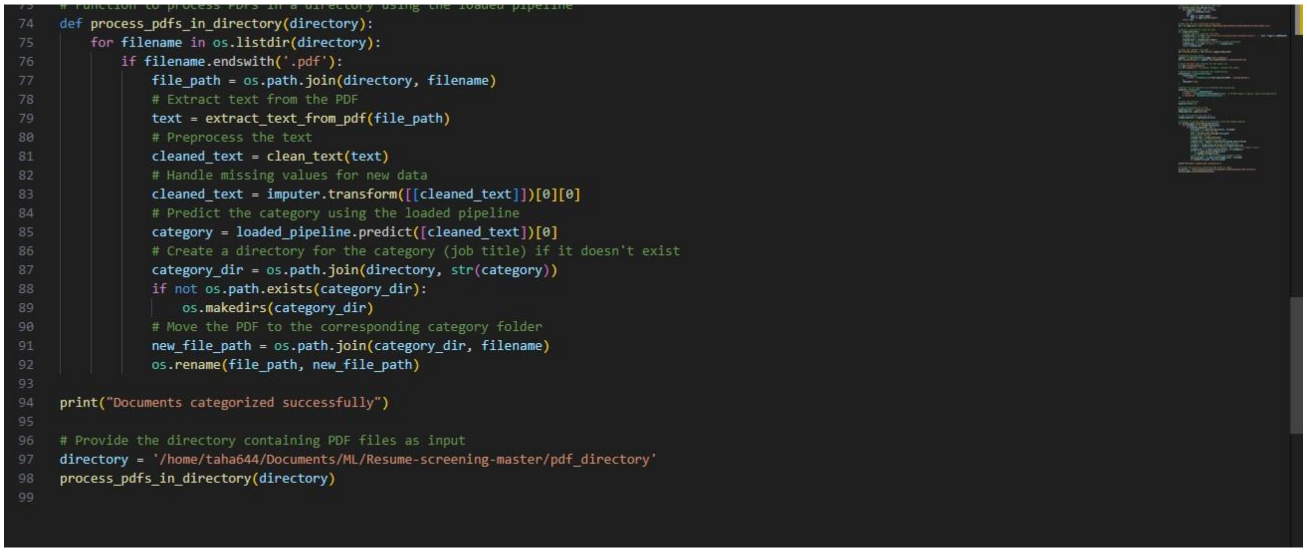
```python
import pandas as pd
import re
import os
from PyPDF2 import PdfReader  # type: ignore
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder, FunctionTransformer
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from joblib import dump, load

# Function to extract text from a PDF file
def extract_text_from_pdf(pdf_file):
    with open(pdf_file, 'rb') as file:
        reader = PdfReader(file)
        text = ''
        for page in reader.pages:
            text += page.extract_text()
    return text

# Read the CSV file containing resume data
df = pd.read_csv("C:/Users/Kainat Ahmed/Downloads/Resume-Screener/Resume-Screener/data.csv")

# Define a function to clean the text
def clean_text(text):
    # Remove specific patterns from text
    cleaned_text = re.sub(r'Page\s+\d+\s+of\s+\d+|Confidential|Header|Footer', '', text, flags=re.IGNORECASE)
```



```python
    return text

# Read the CSV file containing resume data
df = pd.read_csv("C:/Users/Kainat Ahmed/Downloads/Resume-Screener/Resume-Screener/data.csv")

# Define a function to clean the text
def clean_text(text):
    # Remove specific patterns from text
    cleaned_text = re.sub(r'Page\s+\d+\s+of\s+\d+|Confidential|Header|Footer', '', text, flags=re.IGNORECASE)
    # Convert text to lowercase
    cleaned_text = cleaned_text.lower()
    # Remove all non-alphanumeric characters except whitespace
    cleaned_text = re.sub(r'[^\w\s]', '', cleaned_text)
    return cleaned_text

# Clean the 'Resume' text data
df['Cleaned_Resume'] = df['Resume'].apply(clean_text)

# Handling missing values
imputer = SimpleImputer(strategy='most_frequent')
df['Cleaned_Resume'] = imputer.fit_transform(df[['Cleaned_Resume']])

# Split the data into features (X) and labels (y)
X = df[['Cleaned_Resume']]
y = df['Category']  # Assuming 'Category' contains the labels

# Define the column transformer for preprocessing
```

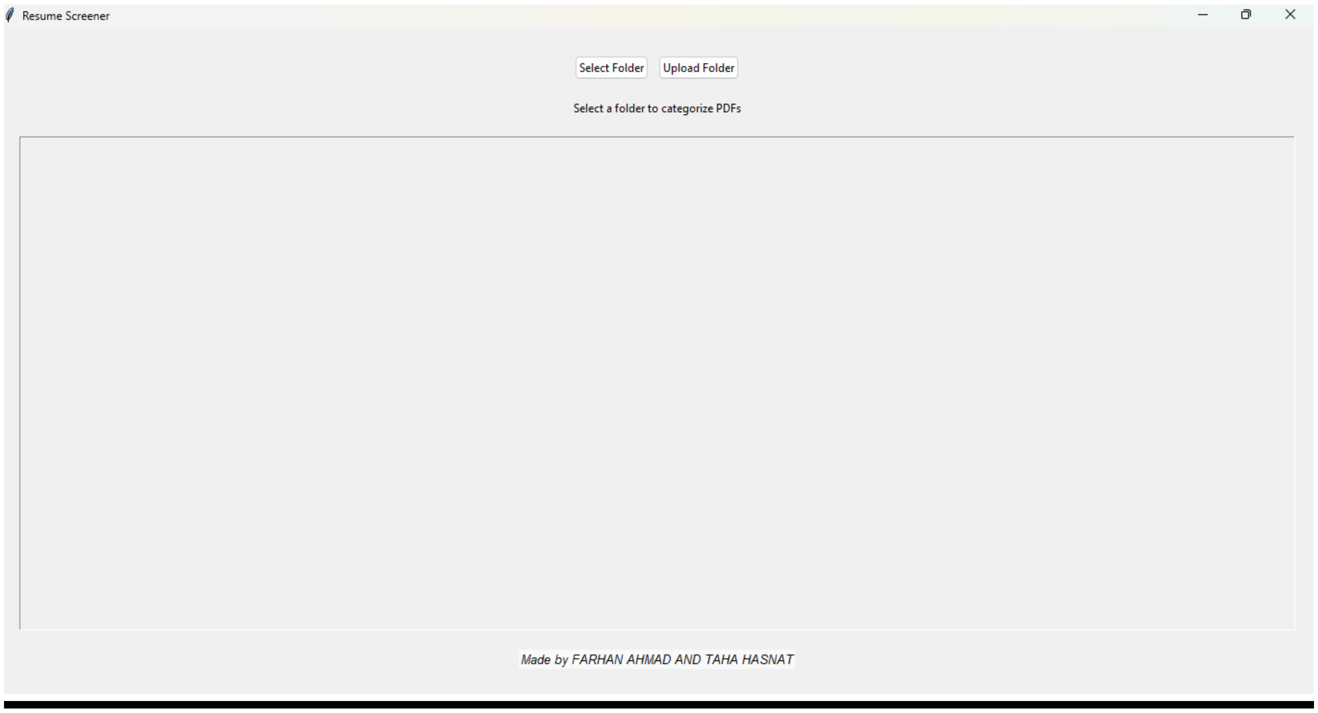


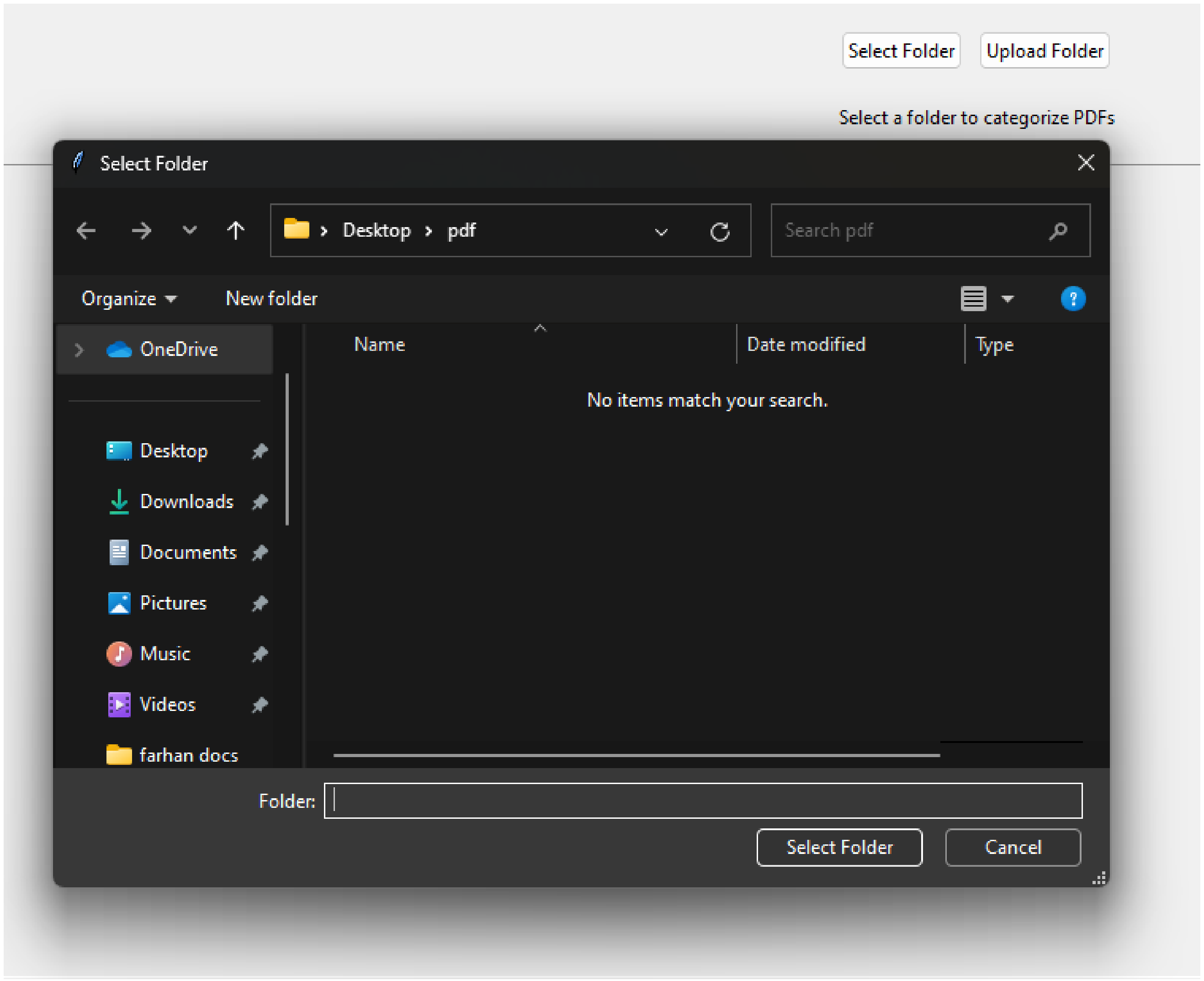
```python
preprocessor = ColumnTransformer(
    transformers=[
        ('tfidf', TfidfVectorizer(max_features=1000), 'Cleaned_Resume'),
    ],
    remainder='drop'
)

# Define the full pipeline with advanced data processing
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('scaler', StandardScaler(with_mean=False)),  # TF-IDF output is sparse, hence with_mean=False
    ('classifier', RandomForestClassifier())
])

# Train the pipeline
pipeline.fit(X, y)

# Save the pipeline to a file
pipeline_file = 'pipeline.joblib'
dump(pipeline, pipeline_file)

# Load the pipeline from the file
loaded_pipeline = load(pipeline_file)

# Function to process PDFs in a directory using the loaded pipeline
def process_pdfs_in_directory(directory):
    for filename in os.listdir(directory):
        if filename.endswith('.pdf'):
            file_path = os.path.join(directory, filename)
            # Extract text from the PDF
```

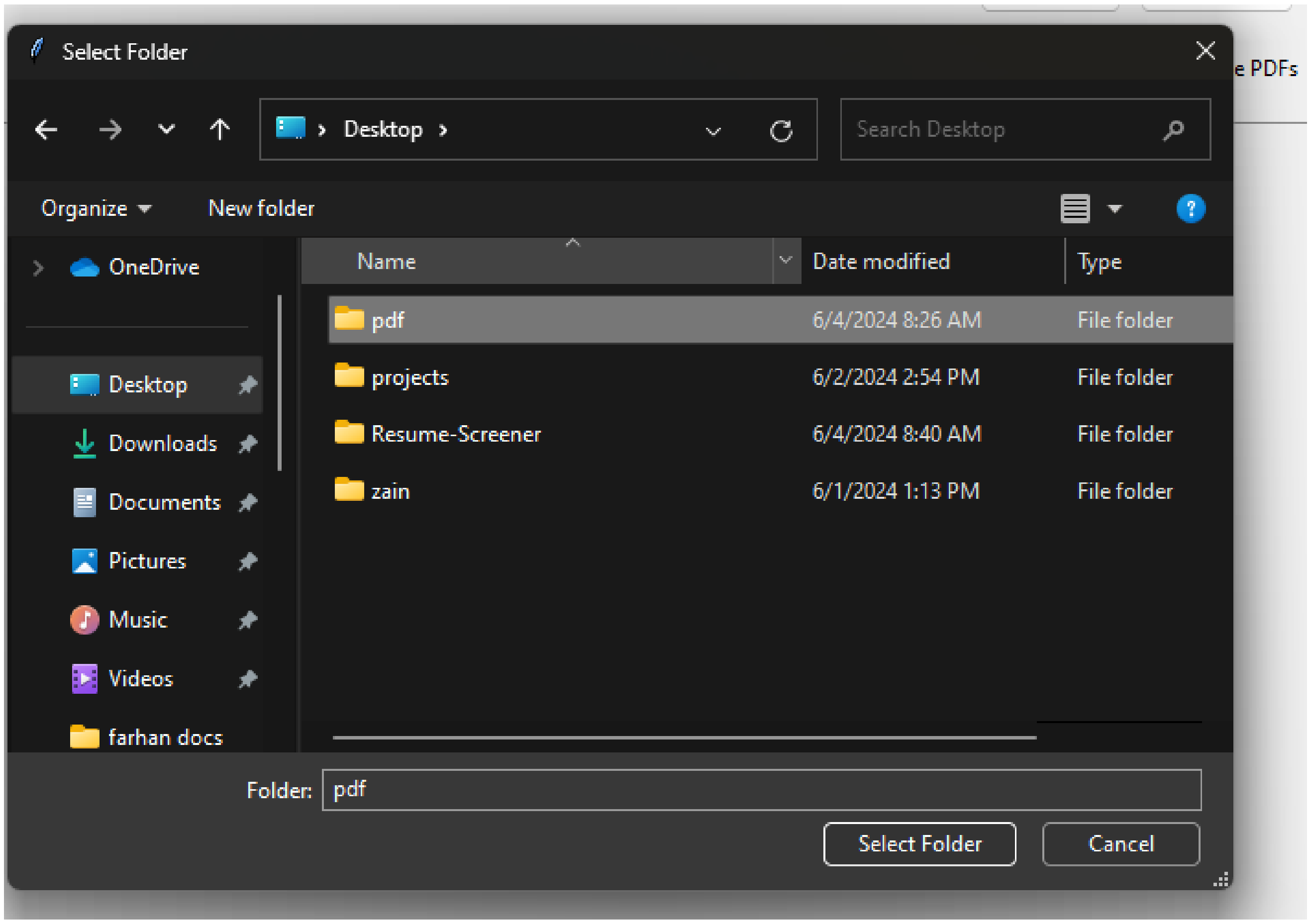


```python
# Function to process PDFs in a directory using the loaded pipeline
def process_pdfs_in_directory(directory):
    for filename in os.listdir(directory):
        if filename.endswith('.pdf'):
            file_path = os.path.join(directory, filename)
            # Extract text from the PDF
            text = extract_text_from_pdf(file_path)
            # Preprocess the text
            cleaned_text = clean_text(text)
            # Handle missing values for new data
            cleaned_text = imputer.transform([[cleaned_text]])[0][0]
            # Predict the category using the loaded pipeline
            category = loaded_pipeline.predict([cleaned_text])[0]
            # Create a directory for the category (job title) if it doesn't exist
            category_dir = os.path.join(directory, str(category))
            if not os.path.exists(category_dir):
                os.makedirs(category_dir)
            # Move the PDF to the corresponding category folder
            new_file_path = os.path.join(category_dir, filename)
            os.rename(file_path, new_file_path)

print("Documents categorized successfully")

# Provide the directory containing PDF files as input
directory = '/home/taha644/Documents/ML/Resume-screening-master/pdf_directory'
process_pdfs_in_directory(directory)
```

OUTPUT:

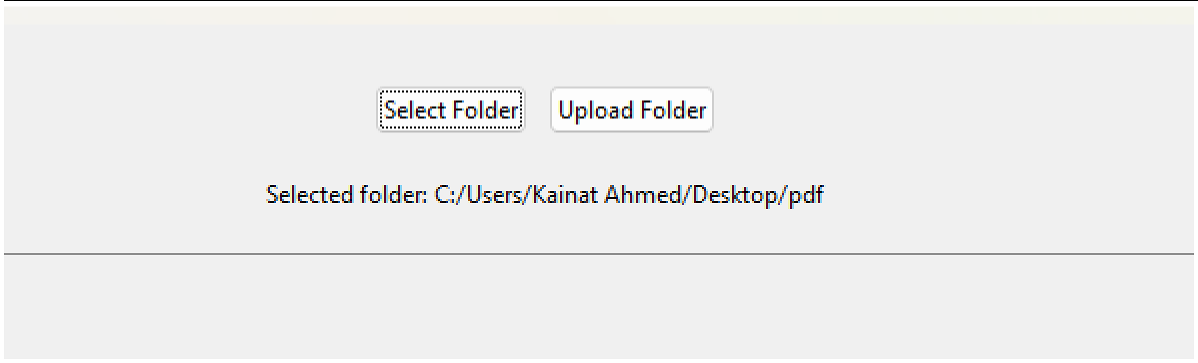

Resume Screener
Select Folder
Upload Folder
Select a folder to categorize PDFs
Made by FARHAN AHMAD AND TAHA HASNAT


Select Folder
Upload Folder
Select a folder to categorize PDFs
Select Folder
Desktop › pdf
Search pdf
Organize
New folder
OneDrive
Desktop
Downloads
Documents
Pictures
Music
Videos
farhan docs
Name
Date modified
Type
No items match your search.
Folder:
Select Folder
Cancel

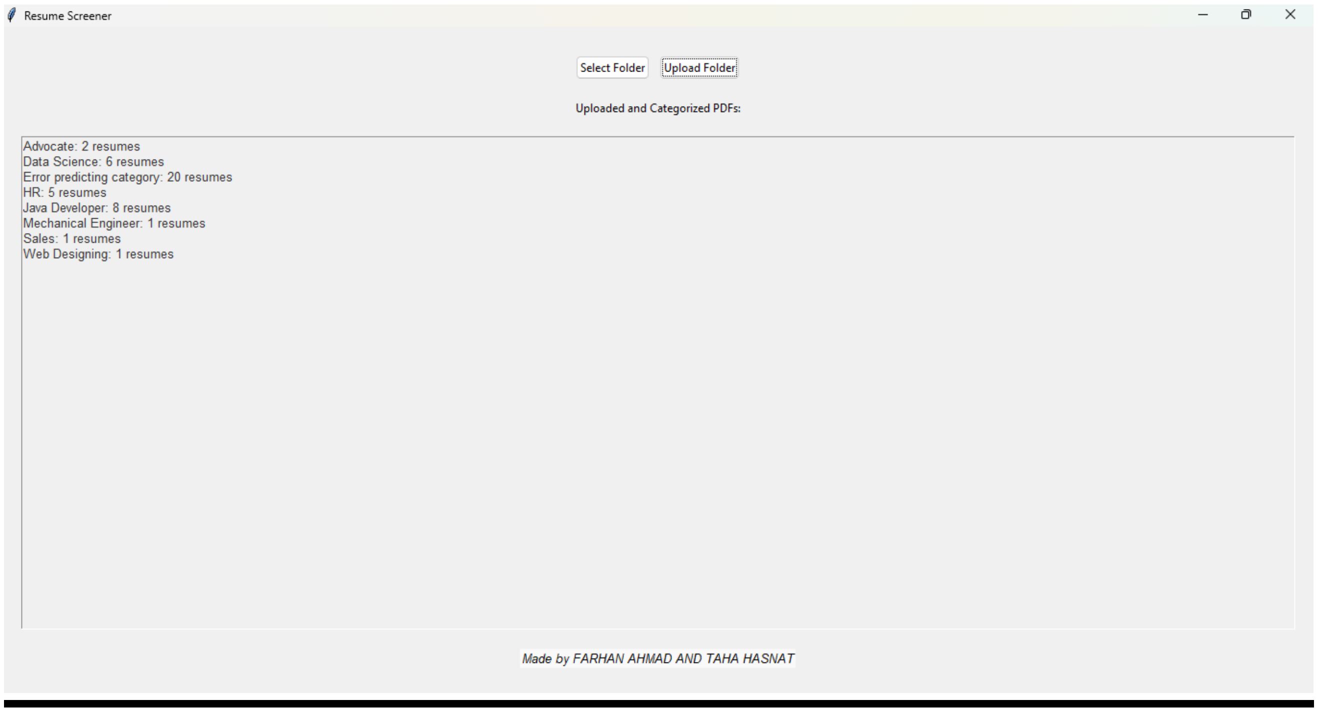


Name	Date modified	Type
pdf	6/4/2024 8:26 AM	File folder
projects	6/2/2024 2:54 PM	File folder
Resume-Screener	6/4/2024 8:40 AM	File folder
zain	6/1/2024 1:13 PM	File folder

Folder: pdf



Selected folder: C:/Users/Kainat Ahmed/Desktop/pdf



```
Advocate: 2 resumes
Data Science: 6 resumes
Error predicting category: 20 resumes
HR: 5 resumes
Java Developer: 8 resumes
Mechanical Engineer: 1 resumes
Sales: 1 resumes
Web Designing: 1 resumes
```

Made by FARHAN AHMAD AND TAHA HASNAT