```python
import pandas as pd
import time
from joblib import Parallel, delayed
from datetime import datetime
import os
import shutil
import logging
from multiprocessing import cpu_count
from openpyxl import Workbook
from openpyxl.styles import PatternFill, Border, Side
from openpyxl.utils import get_column_letter
from concurrent.futures import ThreadPoolExecutor

# Set up logging
logging.basicConfig(filename='process_log.log', level=logging.INFO)

# Start the timer for the total process
start_time = time.time()

# Define the file paths to search
file_paths = [
    r"C:\Users\Stephen\Documents\1. A Work\Test area\Folder 1\large_test_file_1.xlsx",
    r"C:\Users\Stephen\Documents\1. A Work\Test area\Folder 1\large_test_file_2.xlsx",
    r"C:\Users\Stephen\Documents\1. A Work\Test area\Folder 1\large_test_file_3.xlsx",
    # Add more file paths as needed (up to 16 files)
]

# Define the local directory where the files will be copied to
local_directory = r"C:\Users\Stephen\Documents\1. A Work\python-full-3.12.5-windows-x86_64\python-full-3.12.5-windows-x86_64\Scripts"

# Function to copy a single file
def copy_file(file_path, local_directory):
    file_name = os.path.basename(file_path)
    local_file_path = os.path.join(local_directory, file_name)
    shutil.copy2(file_path, local_file_path)
    print(f"Copied {file_name} to {local_file_path}")
    return local_file_path

# Function to copy files in parallel using ThreadPoolExecutor
def copy_files_to_local(shared_file_paths, local_directory, max_workers=8):
    if not os.path.exists(local_directory):
        os.makedirs(local_directory)

    # Use ThreadPoolExecutor to copy files concurrently
    local_file_paths = []
    with ThreadPoolExecutor(max_workers=max_workers) as executor:
        futures = [executor.submit(copy_file, file_path, local_directory) for file_path in shared_file_paths]
        for future in futures:
            local_file_paths.append(future.result())

    return local_file_paths

# Copy files using parallel threads
local_file_paths = copy_files_to_local(file_paths, local_directory, max_workers=8)

# Load the Excel file containing the search terms
search_terms_df = pd.read_excel(r"C:\Users\Stephen\Documents\Extracted_Info.xlsx")

# Extract the second row (index 0 in DataFrame)
row = search_terms_df.iloc[0]

# Function to normalize DOB to a consistent format (dd/mm/yyyy)
def normalize_dob(date_str):
    try:
```

```
            return datetime.strptime(date_str.strip(), "%d/%m/%Y").strftime("%d/%m/%Y")
        except ValueError:
            return date_str.strip()

    # Function to clean strings
    def clean_string(text):
        return text.replace('\n', ' ').replace('\t', ' ').strip()

    # Create a dictionary with the extracted search terms
    new_search_terms = {
        "first_name": clean_string(str(row["First Name"])),
        "last_name": clean_string(str(row["Last Name"])),
        "id": clean_string(str(row["ID 1"])),
        "dob": normalize_dob(clean_string(str(row["DOB"]))),
        "ac_no": clean_string(str(row["AC No."])),
        "phone": clean_string(str(row["Phone"]))
    }

    # Define the search terms by file and sheet
    search_terms_per_file = {
        local_file_paths[0]: {
            "Sheet1": new_search_terms,
            "Sheet2": new_search_terms,
            "Sheet3": new_search_terms,
        },
        local_file_paths[1]: {
            "Sheet1": new_search_terms,
        },
        local_file_paths[2]: {
            "Sheet1": new_search_terms,
        },
    }

    # Function to search for terms in a specific Excel file
    def process_excel_file(file_path, search_terms_per_file):
        all_results = []
        try:
            excel_data = pd.read_excel(file_path, sheet_name=None)
            for sheet_name, sheet_data in excel_data.items():
                cleaned_search_terms = {term_name: clean_string(str(term_value).lower()) for
                term_name, term_value in search_terms_per_file[file_path].get(sheet_name,
                {}).items()}
                headers = list(sheet_data.columns)
                for index, row in sheet_data.iterrows():
                    found_terms = set()
                    matched_values = {}
                    for term_name, term_value_str in cleaned_search_terms.items():
                        if any(term_value_str in clean_string(str(cell).lower()) for cell in
                        row):
                            found_terms.add(term_name)
                            matched_values[term_name] = term_value_str
                    if len(found_terms) >= 2:
                        result = {
                            "Matched Terms": matched_values,
                            "File": file_path,
                            "File Name": os.path.basename(file_path),  # Adding file name
                            here
                            "Sheet": sheet_name,
                            "Row Number": index + 1,
                            "Headers": headers,
                            "Row Data": row.to_dict()
                        }
                        all_results.append(result)
        except Exception as e:
            logging.error(f"Error processing {file_path}: {e}")
        return all_results
```

```
127
128 # Parallel processing to handle files in batches
129 def process_files_in_parallel(file_paths, search_terms_per_file):
130     all_results = Parallel(n_jobs=cpu_count() - 1)(delayed(process_excel_file)(file_path,
        search_terms_per_file) for file_path in file_paths)
131     return [item for sublist in all_results for item in sublist]
132
133 # Execute the parallel processing
134 all_results = process_files_in_parallel(local_file_paths, search_terms_per_file)
135
136 # Prepare for writing to Excel
137 workbook = Workbook()
138 sheet = workbook.active
139 sheet.title = "Results"
140
141 # Add headers for file, sheet, and row
142 sheet.append(["File", "Sheet", "Row", "-------", "Row Data"])
143
144 # Define styles
145 header_fill = PatternFill(start_color="0033A0", end_color="0033A0", fill_type="solid")
    # Dark blue for headers
146 data_fill = PatternFill(start_color="D3D3D3", end_color="D3D3D3", fill_type="solid")
    # Light gray for data
147
148 # Define borders
149 thin_border = Border(left=Side(style='thin'), right=Side(style='thin'), top=Side(style=
    'thin'), bottom=Side(style='thin'))
150
151 # Freeze top row for better navigation
152 sheet.freeze_panes = "A2"
153
154 # Write data with formatting, starting from Column D, without "Headers" or "Row Data"
    labels
155 for idx, result in enumerate(all_results):
156     row_fill = header_fill if idx % 2 == 0 else data_fill
157
158     # Write the actual headers and data without labels
159     row_num = sheet.max_row + 1
160     sheet.append([result["File Name"], result["Sheet"], result["Row Number"], ""] +
        result["Headers"])
161
162     # Add hyperlink to the file path
163     sheet.cell(row=row_num, column=1).hyperlink = result["File"]  # Adds hyperlink to
        cell with the file name
164     sheet.cell(row=row_num, column=1).value = result["File Name"]  # Ensures cell shows
        the file name
165
166     for col_num in range(5, len(result["Headers"]) + 5):
167         cell = sheet.cell(row=row_num, column=col_num)
168         cell.fill = header_fill
169         cell.border = thin_border  # Add borders
170
171     # Write the data row starting from Column D
172     row_num = sheet.max_row + 1
173     sheet.append(["", "", "", ""] + list(result["Row Data"].values()))
174     for col_num in range(5, len(result["Row Data"].values()) + 5):
175         cell = sheet.cell(row=row_num, column=col_num)
176         cell.fill = data_fill
177         cell.border = thin_border  # Add borders
178
179 # Adjust column widths for better readability
180 for col in sheet.columns:
181     max_length = 0
182     column = col[0].column_letter
183     for cell in col:
184         try:
```

```
            if len(str(cell.value)) > max_length:
                max_length = len(cell.value)
        except:
            pass
    adjusted_width = (max_length + 2) if max_length < 30 else 30
    sheet.column_dimensions[column].width = adjusted_width

# Save the workbook
current_time = datetime.now().strftime("%Y-%m-%d %H-%M-%S")
output_file = f"{current_time} Search_results.xlsx"
workbook.save(output_file)

# Generate text file with output file name
def generate_text_file(output_file_name):
    text_file_path = r"C:\Users\Stephen\Documents\1. A
    Work\python-full-3.12.5-windows-x86_64\python-full-3.12.5-windows-x86_64\Scripts\gene
    rated_file_name.txt"
    with open(text_file_path, 'w') as text_file:
        text_file.write(f"{output_file_name}\n")
    print(f"Text file generated at: {text_file_path}")

generate_text_file(output_file)

# End the timer and calculate the total runtime
end_time = time.time()
total_runtime = end_time - start_time

print(f"Total search complete. Results saved to {output_file}")
print(f"Total Runtime: {total_runtime:.2f} seconds")

```