### Main\_Data\_Extraction.py (For the data extraction, you only need to run this code)

This script automates the execution of a series of Python files and then consolidates their output into a single CSV file. First, it imports necessary libraries, sets an address parameter, and specifies a list of Python files to run. It then defines a function, 'run\_python\_file', to execute the files using the 'subprocess' library. The function takes a filename and an optional address parameter, which is required for some files in the list. It then iterates through the list of Python files, executing each one in turn and printing the status of the process. If any file encounters an error, the script stops running the remaining files.

In the second part of the code, it changes the working directory to where the CSV files are located and reads them into pandas data frames. After reading all the CSV files, it concatenates them into a single data frame using an inner join on the "File" column. Finally, it saves the combined dataframe as a new CSV file called "Data.csv".

### 1\_Junk\_file\_removal.py

This code identifies and moves corrupted, encrypted, and similar PDF files to a junk folder. To do this, it first imports necessary libraries for file management and PDF file handling, and then defines three functions to check if a PDF file is corrupted, password-protected, or has more than 80% duplicate pages. After that, it specifies the source folder containing the PDF files and creates a junk folder if it doesn't exist. The script also creates a report file within the junk folder to log the moved files and the reasons for moving them.

The script then iterates through all the files in the source folder, and for each PDF file, it checks the conditions using the defined functions. If any of the conditions are met, the script writes a log message to the report file, explaining the reason for moving the file to the junk folder, and then moves the file to the junk folder. If none of the conditions are met, the script continues to the next file in the folder without taking any action on the current file.

### 2\_Name.py

This code designed to process PDF files within a directory, extract company names from the files, and generate a pandas DataFrame containing the file names and their corresponding company names. To achieve this, the script imports necessary libraries for file management, text extraction from PDF files, and the pre-trained question-answering model from the Transformers library. The script features three main functions: read\_pdf, which utilizes Pdfminer to extract text from a PDF file; extract\_company\_name, which employs a pre-trained question-answering model (specifically, the distilbert-base-cased-distilled-squad model) from the Transformers library to identify the company name within the text; and process\_pdf\_files, responsible for reading the PDF files in a directory, extracting the company names, and constructing a pandas DataFrame.

After specifying the path for the directory containing the PDF files, the script calls the process\_pdf\_files function to create a DataFrame with the extracted company names. It then displays the DataFrame and saves it as a CSV file named 'Name.csv'. The pdfminer library is used to read the PDF files, while the Transformers library's distilbert-base-cased-distilled-squad model is leveraged for question-answering without any data cleaning or preprocessing. The model is prompted with the question "What is the name of the company?" to extract company names from the PDF files' text content.

### Age.py

This Python script searches the web for the age of companies by querying the OpenAI API. First, it imports necessary libraries for configuration parsing, regular expressions, data manipulation, and time management. The script defines several functions: load\_openai\_api\_key to load the API key from a configuration file, get\_founding\_year to query the OpenAI API for the founding year of a given company, rate\_limited\_request to handle rate limit errors from the OpenAI API, get\_company\_age to calculate the age of a company based on the founding year, and get\_founding\_years\_and\_ages to update the pandas DataFrame with company ages.

In the main section of the script, it reads a CSV file containing company names and loads the OpenAI API key. Then, it calls the get\_founding\_years\_and\_ages function to update the DataFrame with the companies' ages. The script uses the rate\_limited\_request function to ensure that the program doesn't exceed the API rate limit. If a company's founding year cannot be determined, the script prints "N/A" for that company.

After obtaining the company ages, the script removes the 'Company' column from the DataFrame, retaining only the 'Age' column. Finally, it saves the resulting DataFrame to a new CSV file named "Age.csv". The script effectively combines data manipulation and API querying to determine the age of each company in the input file and create an output file with the relevant information.

### Clinical\_Phase.py

This code designed to analyze PDF files in a directory, extract the text content, and determine the clinical stage of a company based on the text. To achieve this, the script imports necessary libraries for file management, text extraction from PDF files, and the pre-trained question-answering model from the Transformers library. The script defines several functions: extract\_text\_from\_pdf to read the text content of a PDF file using PyPDF2, find\_clinical\_stage to query the pre-trained question-answering model and extract the clinical stage from the text, get\_clinical\_stage\_integer to convert the answer text to an integer representing the clinical stage, and process\_pdf\_files to read the PDF files in a directory and build a pandas DataFrame containing the file names and clinical stages.

In the main function, the script sets the path of the directory containing the PDF files and calls the process\_pdf\_files function to create a DataFrame with the extracted clinical stages. For each PDF file, it reads the text content, checks if the content is not empty, and then queries the question-answering model with the question "In which clinical stage phase is the company?". It then converts the answer text to an integer representing the clinical stage and adds the information to the DataFrame. If the text extraction from the PDF fails, the script logs a message and sets the clinical stage as "NA". Finally, the resulting DataFrame is saved as a CSV file named "Clinical.csv".

### Computationality.py

This code designed to assess how computational a company's approach is by analyzing PDF files in a directory, extracting their text content, and calculating the semantic similarity between the text and a reference text. The script uses the PyPDF2 library to handle PDF files, the Transformers library for pre-trained natural language processing (NLP) models and tokenizers, and the Pandas library for data manipulation. It defines several functions: read\_pdf\_files to read the text content of PDF files in a directory, encode\_text to convert text into numerical embeddings using a pre-trained NLP model and tokenizer, and calculate\_similarity\_scores to compute the similarity scores between the file texts and the reference text.

In the main section of the script, it sets the path of the directory containing the PDF files and the reference text, which describes a computational approach to biology. The script loads a pre-trained NLP model and tokenizer—specifically, the "sentence-transformers/stsb-roberta-base" model. It then reads the PDF files, calculates similarity scores between the file texts and the reference text using the pre-trained NLP model, and normalizes the scores to a scale of 0 to 10.

The script creates a pandas DataFrame from the normalized scores dictionary, removes the ".pdf" extension from the file names, and rounds the similarity scores to two decimal places. Finally, it saves the resulting DataFrame as a CSV file named "Computationality.csv". This script effectively combines PDF file processing, pre-trained NLP models, and data manipulation to evaluate how computational each company's approach is, based on the content of their PDF files.

### Diag\_Treatment.py

This code designed to analyze PDF files in a directory, extract their text content, and classify the content into either "Disease Diagnosis" or "Disease Treatment" categories based on the semantic similarity of the text to a set of predefined phrases. The script uses the PyPDF2 library for handling PDF files, the sentence-transformers library for pre-trained natural language processing (NLP) models, and the Pandas library for data manipulation. It defines several functions: pdf\_to\_text to read the text content of PDF files, and process\_text to classify the text into categories using the pre-trained NLP model.

In the main section of the script, it sets the path of the directory containing the PDF files, loads a pre-trained sentence-transformer model (specifically, 'paraphrase-MiniLM-L6-v2'), and defines the categories along with their related phrases. The script then converts the phrases into embeddings using the pre-trained NLP model. It reads the PDF files, processes the text using the model and the category embeddings, and appends the results to a list. Each PDF file is classified as either "Diagnosis" or "Treatment" based on the majority count of chunks belonging to each category.

The script creates a pandas DataFrame with the results, removing the ".pdf" extension from the file names, and saves the DataFrame as a CSV file named "Diag\_Treat.csv". This script effectively combines PDF file processing, pre-trained NLP models, and data manipulation to classify the content of PDF files as either "Disease Diagnosis" or "Disease Treatment" based on their semantic similarity to a set of predefined phrases.

### Issue\_category\_finder.py

This code consists of two main parts. The first part is a script that uses the transformers library to extract the main disease mentioned in a collection of PDF files. It first imports necessary libraries and defines several functions. The 'extract\_pdf\_text' function extracts text from a PDF file using the PyPDF2 library. The 'split\_into\_chunks' function divides the text into chunks of a specified size. The 'find\_diseases' function processes these chunks using a pre-trained question-answering model from the transformers library to identify the main disease mentioned in each chunk. The 'process\_files' function processes all the PDF files in a specified folder, calling the previous functions to extract text, split it into chunks, and find diseases. Finally, it creates a DataFrame with the extracted data and saves it as a CSV file.

The second part of the code uses the OpenAI GPT-3 API to classify the main disease keywords extracted from the PDF files into related areas. It first imports necessary libraries and defines more functions. The 'load\_openai\_api\_key' function retrieves the API key from a configuration file. The 'find\_matching\_area' function identifies the matching area for a given keyword based on a predefined list of areas. The 'get\_related\_area' function sends the keywords to the GPT-3 API and receives an answer, which is then processed by the 'find\_matching\_area' function to determine the related area. The main function reads the CSV file created in the first part, adds a new column for the issue, and applies the 'get\_related\_area' function to each keyword. It then drops the keyword column, saves the updated DataFrame as a new CSV file, and prints the DataFrame.

### Location.py

This code aims to determine the location of companies and classify them into specific geographical areas. The code first imports necessary libraries and defines several functions. The 'load\_openai\_api\_key' function retrieves the API key from a configuration file. The 'get\_company\_location' function queries the OpenAI GPT-3 API to get the location of a company in the format "Company Name, Location". The 'get\_company\_locations' function iterates through the rows of a DataFrame containing company names, and for each company, it calls the 'get\_company\_location' function to retrieve the location. If the API rate limit is reached, the script waits for a specified amount of time before retrying the request.

The second part of the code defines a function 'get\_location\_area' that classifies the company locations into specific areas based on predefined lists of keywords for East USA, West USA, and Heartland USA. It reads a CSV file containing the company locations, processes the location column to determine the location area using the 'get\_location\_area' function, and creates a new DataFrame with the original file names and the location areas. Finally, it saves the new DataFrame as a CSV file. In this code, the question-answering package used is OpenAI GPT-3 API, which is utilized to determine the company's location.

### Method.py

This script reads PDF files from the specified folder and classifies each file based on the content into one of five categories: Drug Discovery, Genomics or Gene Editing, Immunotherapy or Cell Engineering, Microbiome, and Computational Biology and AI. The classification is performed by counting the occurrences of specific keywords related to each category in the text of each PDF file. The script uses the PyPDF2 library to handle the PDF files, the re library for keyword matching, and the pandas library for creating and exporting data as a CSV file. It defines two functions: get\_text\_from\_pdf to extract text from a PDF file, and main\_method\_area to determine the main method category based on the keyword counts in the text.

In the main section of the script, it sets the path of the directory containing the PDF files and initializes an empty list to store the results. It also defines the keywords for each category. The script then iterates through the files in the directory, extracts the text from each PDF file using the get\_text\_from\_pdf function, and classifies the main method area using the main\_method\_area function. The results are stored in the 'data' list as dictionaries containing the file name and the method area.

Finally, the script creates a pandas DataFrame from the 'data' list, removing the ".pdf" extension from the file names, and saves the DataFrame as a CSV file named "Method.csv". This script effectively combines PDF file processing, keyword matching, and data manipulation to classify the content of PDF files into one of the predefined method categories based on the occurrence of specific keywords in the text.