Contents

[pdf\_module 3](#_Toc171863348)

[1. Function: read\_text 3](#_Toc171863349)

[2. Function: is\_Searchable 3](#_Toc171863350)

[3. Function: is\_valid\_pdf\_link 4](#_Toc171863351)

[4. Function: download\_pdf 4](#_Toc171863352)

[5. Function: process\_pdf 5](#_Toc171863353)

[6. Function: extract\_text\_from\_pdf\_Production 5](#_Toc171863354)

[7. Function: get\_links 6](#_Toc171863355)

[8. Function: get\_all\_files 6](#_Toc171863356)

[9. Function: extract\_text\_from\_pdf 7](#_Toc171863357)

[10. Function: data\_extract3 7](#_Toc171863358)

[txtFileManagement 9](#_Toc171863359)

[1. Function: remove\_o() 9](#_Toc171863360)

[2. Function: export\_id() 9](#_Toc171863361)

[3. Function: get\_ids\_from\_debug() 10](#_Toc171863362)

[4. Function: update\_classification() 10](#_Toc171863363)

[5. Function: save\_csv\_to\_files() 11](#_Toc171863364)

[6. Function: join\_Files() 11](#_Toc171863365)

[7. Function: check\_csv() 12](#_Toc171863366)

[8. Function: remove\_duplicates() 12](#_Toc171863367)

[9. Example Usage 13](#_Toc171863368)

[Model 16](#_Toc171863369)

[1. Classes 16](#_Toc171863370)

[**TextPreprocessor** 16](#_Toc171863371)

[2. TrainingModels 17](#_Toc171863372)

[**Usage** 20](#_Toc171863373)

[**Preprocessing Text** 20](#_Toc171863374)

[**Training Models** 20](#_Toc171863375)

[**Predicting Classes** 21](#_Toc171863376)

[FastAPI Application and Routes 22](#_Toc171863377)

[**TrainingModels Initialization** 23](#_Toc171863378)

[**FastAPI Application Setup** 23](#_Toc171863379)

# pdf\_module

The pdf\_module.py is a comprehensive utility module designed for handling various PDF-related tasks, including reading, validating, downloading, processing, and extracting text from PDF files. The module leverages several libraries like requests, pdfplumber, fitz (PyMuPDF), and pytesseract to provide robust solutions for working with PDFs. Key functionalities include determining if PDFs are searchable, downloading PDFs from valid links, extracting text using OCR when necessary, and managing PDF links stored in CSV files. This module is particularly useful for automating the extraction of textual data from large collections of PDF documents, making it a valuable tool for data processing and analysis workflows.

## Function: read\_text

**Description**: This function reads the content of a text file from a given file path and returns the text as a string. If an error occurs during the reading process, it prints an error message and returns an empty string.

**Parameters**:

* file\_path (str): The path to the text file.

**Return Type**:

* str: The content of the text file or an empty string if an error occurs.

**Sample Use Case**:

python

file\_path = 'example.txt'

text = read\_text(file\_path)

print(text)

## Function: is\_Searchable

**Description**: This function checks whether a PDF file contains searchable text based on a given threshold. It considers the top and bottom margins to exclude header and footer content.

**Parameters**:

* pdf\_path (str): The path to the PDF file.
* top\_margin (float): The margin from the top of the page to exclude from text extraction (default: constants.top\_margin).
* bottom\_margin (float): The margin from the bottom of the page to exclude from text extraction (default: constants.bottom\_margin).
* thresh (int): The threshold percentage to consider a PDF as searchable (default: 30).

**Return Type**:

* bool: True if the PDF is considered searchable, False otherwise.

**Sample Use Case**:

python

pdf\_path = 'example.pdf'

is\_searchable = is\_Searchable(pdf\_path, top\_margin=0.1, bottom\_margin=0.1, thresh=30)

print(is\_searchable)

## Function: is\_valid\_pdf\_link

**Description**: This function checks if a given URL is a valid PDF link, specifically for URLs starting with "<http://download.siliconexpert.com/pdfs>" or "<https://download.siliconexpert.com/pdfs>" and ending with ".pdf".

**Parameters**:

* url (str): The URL to be checked.

**Return Type**:

* bool: True if the URL is a valid PDF link, False otherwise.

**Sample Use Case**:

python

url = 'https://download.siliconexpert.com/pdfs/example.pdf'

is\_valid = is\_valid\_pdf\_link(url)

print(is\_valid)

## Function: download\_pdf

**Description**: This function downloads a PDF from a given URL and saves it to a specified path if the URL is valid.

**Parameters**:

* url (str): The URL of the PDF to be downloaded.
* save\_path (str): The file path where the PDF should be saved.

**Return Type**:

* bool: True if the PDF was successfully downloaded, False otherwise.

**Sample Use Case**:

python

url = 'https://download.siliconexpert.com/pdfs/example.pdf'

save\_path = 'example.pdf'

download\_status = download\_pdf(url, save\_path)

print(download\_status)

## Function: process\_pdf

**Description**: This function processes a PDF byte stream by converting it to images and performing OCR to extract text from the first few pages.

**Parameters**:

* pdf\_bytes (bytes): The PDF byte stream to be processed.
* max\_pages (int): The maximum number of pages to process (default: 3).

**Return Type**:

* dict: A dictionary with keys text (str) and status (int). The text key contains the extracted text, and the status key indicates success (1) or failure (0).

**Sample Use Case**:

python

pdf\_url = 'https://example.com/sample.pdf'

response = requests.get(pdf\_url)

pdf\_bytes = response.content

result = process\_pdf(pdf\_bytes, max\_pages=3)

print(result)

## Function: extract\_text\_from\_pdf\_Production

**Description**: This function extracts text from a PDF file located at a URL, using specific margins to exclude header and footer content. It uses OCR if the initial text extraction is insufficient.

**Parameters**:

* pdf\_path (str): The URL of the PDF to extract text from.
* top\_margin (float): The margin from the top of the page to exclude from text extraction (default: constants.top\_margin).
* bottom\_margin (float): The margin from the bottom of the page to exclude from text extraction (default: constants.bottom\_margin).

**Return Type**:

* str: The extracted text from the PDF, or an empty string if extraction fails.

**Sample Use Case**:

python

pdf\_url = 'https://example.com/sample.pdf'

extracted\_text = extract\_text\_from\_pdf\_Production(pdf\_url)

print(extracted\_text)

## Function: get\_links

**Description**: This function reads a CSV file and returns a DataFrame with valid PDF links after filtering out invalid ones.

**Parameters**:

* file\_path (str): The path to the CSV file containing PDF links (default: constants.not\_datasheet\_link\_file).

**Return Type**:

* pd.DataFrame: A DataFrame with valid PDF links after filtering out invalid ones.

**Sample Use Case**:

python

csv\_file\_path = 'pdf\_links.csv'

df = get\_links(file\_path=csv\_file\_path)

print(df.head())

## Function: get\_all\_files

**Description**: This function consolidates PDF links from two CSV files, removes duplicates, and returns a combined DataFrame.

**Parameters**:

* pdf\_path (str): The path to the CSV file with non-datasheet PDF links (default: constants.not\_datasheet\_link\_file).
* datasheet\_path (str): The path to the CSV file with datasheet PDF links (default: constants.datasheet\_link\_file).
* use\_Exist (bool): Whether to use an existing consolidated file if available (default: True).

**Return Type**:

* pd.DataFrame: A DataFrame containing all consolidated PDF links after removing duplicates.

**Sample Use Case**:

python

all\_files\_df = get\_all\_files(pdf\_path='not\_datasheet\_links.csv', datasheet\_path='datasheet\_links.csv', use\_Exist=True)

print(all\_files\_df.info())

## Function: extract\_text\_from\_pdf

**Description**: This function extracts text from a PDF file and saves it to a specified text file path. It uses specific margins to exclude header and footer content.

**Parameters**:

* pdf\_path (bytes): The byte stream of the PDF file.
* txt\_path (str): The file path where the extracted text should be saved.
* top\_margin (float): The margin from the top of the page to exclude from text extraction (default: constants.top\_margin).
* bottom\_margin (float): The margin from the bottom of the page to exclude from text extraction (default: constants.bottom\_margin).

**Return Type**:

* bool: True if the text was successfully extracted and saved, False otherwise.

**Sample Use Case**:

python

pdf\_path = 'example.pdf'

txt\_path = 'example.txt'

extraction\_status = extract\_text\_from\_pdf(pdf\_path, txt\_path)

print(extraction\_status)

## Function: data\_extract3

**Description**: This function extracts data from a collection of PDF files, processes them, and stores the extracted text in a specified directory.

**Parameters**:

* dataLabel\_List (list): List of data labels to be processed (default: []).
* exclude\_list (list): List of project labels to exclude from processing (default: ['Delete\_HTML', 'In\_Complete\_DS']).
* start (int): The starting index for processing (default: 0).
* finish (int): The ending index for processing (default: 10000000).

**Return Type**:

* None: This function does not return a value.

**Sample Use Case**:

python

extracted\_data = data\_extract3(dataLabel\_List=['Label1', 'Label2'], exclude\_list=['Delete\_HTML', 'In\_Complete\_DS'], start=0, finish=100)

print("Data extraction complete.")

# txtFileManagement

This module provides utility functions for handling and processing CSV files and text files in a specified directory. The functions included cover a variety of tasks, such as renaming files, exporting IDs to a CSV file, updating classification data, merging CSV files, and removing duplicates. This module is useful for organizing and managing large datasets, particularly for tasks related to text extraction, data classification, and CSV file manipulation. It leverages libraries like os for file operations and pandas for data manipulation, making it a versatile tool for data processing workflows.

## Function: remove\_o()

**Description:**

Renames files in the specified directory by removing the '.0' from the filenames.

**Parameters:**

* directory (str): The directory containing the files to be renamed. Default is r"data\text\_files".

**Return Type:**

* None

**Example:**

python

remove\_o()

## Function: export\_id()

**Description:**

Exports the IDs (filenames without extensions) of all files in the specified directory to a CSV file.

**Parameters:**

* directory (str): The directory containing the files. Default is r"data\text\_files".
* filename (str): The name of the CSV file to save the IDs. Default is 'all\_Text.csv'.

**Return Type:**

* None

**Example:**

python

export\_id()

## Function: get\_ids\_from\_debug()

**Description:**

Extracts the 'pdf\_url' column from a CSV file and saves it to a new CSV file.

**Parameters:**

* source\_path (str): The path of the source CSV file. Default is r'C:\Users\161070\Downloads\FULL\_DATA\_out.csv'.
* dest\_path (str): The path of the destination CSV file. Default is r'C:\Users\161070\OneDrive - Arrow Electronics, Inc\Desktop\ids\_file.csv'.

**Return Type:**

* None

**Example:**

python

get\_ids\_from\_debug()

## Function: update\_classification()

**Description:**

Updates the classification of documents by merging data from two CSV files based on 'PDF\_ID'.

**Parameters:**

* id\_source (str): The path of the CSV file containing the IDs. Default is r'C:\Users\161070\OneDrive - Arrow Electronics, Inc\Desktop\datasheet\_filters.csv'.
* data\_classification (str): The path of the CSV file containing the classification data. Default is r'C:\Users\161070\OneDrive - Arrow Electronics, Inc\Desktop\Document Classification\data\Log\debug\_data\_out.csv'.

**Return Type:**

* None

**Example:**

python

update\_classification()

## Function: save\_csv\_to\_files()

**Description:**

Reads a CSV file and saves each row's text content to a separate text file named after the 'PDF\_ID'.

**Parameters:**

* path (str): The path of the CSV file to read. Default is r"C:\Users\161070\Downloads\last200KSupplierPDF2\_text\_part\_8 (1).csv".
* dir (str): The directory where the text files will be saved. Default is r'C:\Users\161070\OneDrive - Arrow Electronics, Inc\Desktop\Document Classification\data\text\_files'.

**Return Type:**

* None

**Example:**

python

save\_csv\_to\_files()

## Function: join\_Files()

**Description:**

Merges two CSV files based on the 'PDF\_URL' column and saves the result to a new CSV file.

**Parameters:**

* file1 (str): The path of the first CSV file to merge. Default is r"C:\Users\161070\OneDrive - Arrow Electronics, Inc\Desktop\datasheet\_all\TABLE\_EXPORT\_DATA\_1.csv".
* file2 (str): The path of the second CSV file to merge. Default is r"C:\Users\161070\Downloads\FULL\_DATA\_out.csv".
* res (str): The path of the resulting CSV file. Default is r"C:\Users\161070\Downloads\FULL\_DATA\_out2.csv".

**Return Type:**

* None

**Example:**

python

join\_Files()

## Function: check\_csv()

**Description:**

Prints the columns, length, and value counts of the 'classification' and 'PROJECT' columns of a CSV file.

**Parameters:**

* path (str): The path of the CSV file to check. Default is r'C:\Users\161070\OneDrive - Arrow Electronics, Inc\Desktop\Document Classification\data\Log\debug\_data\_out\_last.csv'.

**Return Type:**

* None

**Example:**

python

check\_csv()

## Function: remove\_duplicates()

**Description:**

Removes duplicate rows from a CSV file based on 'PDF\_URL' and 'PDF\_ID' columns.

**Parameters:**

* file (str): The path of the CSV file to remove duplicates from. Default is r'C:\Users\161070\OneDrive - Arrow Electronics, Inc\Desktop\Document Classification\data\Log\debug\_data\_out5\_.csv'.

**Return Type:**

* None

**Example:**

python

remove\_duplicates()

## Example Usage

#### Example 1: Full Data Processing Workflow

This example demonstrates a complete workflow of processing text files, exporting IDs, updating classifications, and checking the final CSV file.

python

# Step 1: Rename files in the directory by removing '.0'

remove\_o(directory=r"data\text\_files")

# Step 2: Export IDs from filenames to a CSV file

export\_id(directory=r"data\text\_files", filename='extracted\_ids.csv')

# Step 3: Extract PDF URLs from a debug CSV file and save to a new file

get\_ids\_from\_debug(source\_path=r'C:\Users\161070\Downloads\FULL\_DATA\_out.csv',

dest\_path=r'C:\Users\161070\OneDrive - Arrow Electronics, Inc\Desktop\ids\_file.csv')

# Step 4: Update the classification of documents using the exported IDs and classification data

update\_classification(id\_source=r'C:\Users\161070\OneDrive - Arrow Electronics, Inc\Desktop\datasheet\_filters.csv',

data\_classification=r'C:\Users\161070\OneDrive - Arrow Electronics, Inc\Desktop\Document Classification\data\Log\debug\_data\_out.csv')

# Step 5: Save text content from a CSV file to individual text files

save\_csv\_to\_files(path=r"C:\Users\161070\Downloads\last200KSupplierPDF2\_text\_part\_8 (1).csv",

dir=r'C:\Users\161070\OneDrive - Arrow Electronics, Inc\Desktop\Document Classification\data\text\_files')

# Step 6: Check the content and structure of the final CSV file

check\_csv(path=r'C:\Users\161070\OneDrive - Arrow Electronics, Inc\Desktop\Document Classification\data\Log\debug\_data\_out\_last.csv')

#### Example 2: Conditional File Processing

This example processes files based on a condition, such as only processing files with specific IDs.

python

Copy code

# Step 1: Filter files based on a specific condition

directory = r"data\text\_files"

filtered\_files = [f for f in os.listdir(directory) if "specific\_id" in f]

# Step 2: Rename the filtered files

for file in filtered\_files:

if file.endswith('.0.txt'):

new\_filename = file.replace('.0', '')

try:

os.rename(os.path.join(directory, file), os.path.join(directory, new\_filename))

print(f"Renamed {file} to {new\_filename}")

except Exception as e:

print(f"Error renaming {file} to {new\_filename}: {e}")

# Step 3: Export IDs of the filtered files to a CSV file

filtered\_ids = [f.split('.')[0] for f in filtered\_files]

with open('filtered\_ids.csv', mode='w', newline='') as file:

writer = csv.writer(file)

for row in filtered\_ids:

writer.writerow([row])

#### Example 3: Error Handling and Logging

This example incorporates error handling and logging to ensure robust file processing.

python

Copy code

import logging

# Configure logging

logging.basicConfig(filename='file\_processing.log', level=logging.INFO, format='%(asctime)s:%(levelname)s:%(message)s')

def safe\_remove\_o(directory=r"data\text\_files"):

try:

remove\_o(directory=directory)

logging.info(f"Successfully renamed files in directory: {directory}")

except Exception as e:

logging.error(f"Error processing directory {directory}: {e}")

def safe\_export\_id(directory=r"data\text\_files", filename='all\_Text.csv'):

try:

export\_id(directory=directory, filename=filename)

logging.info(f"Successfully exported IDs to {filename}")

except Exception as e:

logging.error(f"Error exporting IDs from directory {directory} to {filename}: {e}")

def safe\_update\_classification(id\_source, data\_classification):

try:

update\_classification(id\_source=id\_source, data\_classification=data\_classification)

logging.info(f"Successfully updated classification using {id\_source} and {data\_classification}")

except Exception as e:

logging.error(f"Error updating classification with {id\_source} and {data\_classification}: {e}")

# Example usage

safe\_remove\_o()

safe\_export\_id()

safe\_update\_classification(id\_source='all\_ids\_combined.csv', data\_classification=r'C:\Users\161070\OneDrive - Arrow Electronics, Inc\Desktop\Document Classification\data\Log\debug\_data\_out.csv')

# Model

## Classes

**TextPreprocessor**

This class is responsible for text preprocessing, including tokenization, stopword removal, lemmatization, stemming, and removal of URLs, numbers, and emojis.

**Attributes:**

* stop\_words: Set of English stopwords.
* lemmatizer: Lemmatizer object for word lemmatization.
* stemmer: Stemmer object for word stemming.
* n\_jobs: Number of parallel jobs for preprocessing.
* emoji\_pattern: Regular expression pattern for emojis.
* url\_pattern: Regular expression pattern for URLs.
* html\_tags\_pattern: Regular expression pattern for HTML tags.

**Methods:**

* fit(X, y=None): Fits the transformer; does nothing in this case.
* transform(X, y=None): Applies preprocessing to the input text.
* \_preprocess(text): Internal method to preprocess a single text string.

**Methods Docs:**

**\_\_init\_\_(self, n\_jobs=-1)**

* **Parameters**:
  + n\_jobs (int): The number of jobs to run in parallel. Default is -1, which means using all processors.
* **Return Type**: None
* **Description**: Initializes the text preprocessor with stopwords, lemmatizer, stemmer, and patterns for emojis, URLs, and HTML tags.

**fit(self, X, y=None)**

* **Parameters**:
  + X (array-like): Input data.
  + y (array-like, optional): Target data.
* **Return Type**: self
* **Description**: This method doesn't perform any fitting but is required by scikit-learn's transformer interface.

**transform(self, X, y=None)**

* **Parameters**:
  + X (array-like): Input data.
  + y (array-like, optional): Target data.
* **Return Type**: list of str
* **Description**: Applies the \_preprocess method to each element in X using parallel processing.

**\_preprocess(self, text)**

* **Parameters**:
  + text (str): Input text.
* **Return Type**: str
* **Description**: Cleans and preprocesses the input text by removing URLs, numbers, punctuation, emojis, and HTML tags, and by applying lemmatization and stopword removal.

## TrainingModels

This class handles the training and evaluation of multiple classifiers on preprocessed text data.

**Attributes:**

* train\_tfidf\_vectorizer: TF-IDF Vectorizer object for text vectorization.
* classifiers: List of classifier tuples (name, classifier).
* data: DataFrame for storing training data.
* label\_encoder: Label encoder for target labels.
* tfidf\_Params: Parameters for TF-IDF vectorization.
* Pipelines: Dictionary to store trained pipelines.
* selected\_Labels: List of selected labels for classification.
* PreLoadText: Flag to indicate if text data should be preloaded.
* set\_preload\_text\_: Internal flag for text preloading.

**Methods:**

* set\_preload\_text(value=True): Sets the preload text flag.
* label\_other(x, labels): Labels elements not in the provided list as 'Others'.
* Load\_data(): Loads data for training.
* load\_Text(): Loads text data from PDF files.
* analyze(): Analyzes the dataset.
* train\_multi\_level(Classes\_sample, selected\_classes, Model\_name): Trains multiple classifiers on hierarchical classes.
* load\_train\_test\_data(): Loads training and test data.
* train(): Trains the classifiers on the dataset.
* load\_models(date): Loads pre-trained models.
* print\_model(name, model): Prints model parameters and features.
* preprocess(text): Preprocesses input text.
* predict\_pdf(url): Predicts the class of a PDF document from its URL.
* predict\_Files(df): Predicts the class of multiple PDF documents from a DataFrame of URLs.
* predict(text\_list): Predicts the class of input text.
* expand\_Models(): Expands the models by loading pre-trained ones and printing their parameters.

**Methods Docs:**

**\_\_init\_\_(self)**

* **Parameters**: None
* **Return Type**: None
* **Description**: Initializes the training models, including setting up the TF-IDF vectorizer and classifiers.

**set\_preload\_text(self, value=True)**

* **Parameters**:
  + value (bool): Flag to set the preload text. Default is True.
* **Return Type**: None
* **Description**: Sets the set\_preload\_text\_ attribute to determine whether to preload text.

**label\_other(self, x, labels)**

* **Parameters**:
  + x (str): Input label.
  + labels (list of str): List of selected labels.
* **Return Type**: str
* **Description**: Labels the input x as 'Others' if it is not in the provided list of labels.

**Load\_data(self)**

* **Parameters**: None
* **Return Type**: None
* **Description**: Loads and preprocesses the data from the file system.

**load\_Text(self)**

* **Parameters**: None
* **Return Type**: None
* **Description**: Loads the text data for each PDF ID.

**analyze(self)**

* **Parameters**: None
* **Return Type**: None
* **Description**: Analyzes the data by printing information and value counts.

**train\_multi\_level(self, Classes\_sample, selected\_classes, Model\_name)**

* **Parameters**:
  + Classes\_sample (list of str): List of class samples.
  + selected\_classes (list of str): List of selected classes.
  + Model\_name (str): Model name.
* **Return Type**: None
* **Description**: Trains multi-level classifiers on the provided class samples.

**load\_train\_test\_data(self)**

* **Parameters**: None
* **Return Type**: tuple (pd.DataFrame, pd.DataFrame, pd.Series, pd.Series)
* **Description**: Loads and splits the data into training and testing sets.

**train(self)**

* **Parameters**: None
* **Return Type**: None
* **Description**: Trains and evaluates the classifiers on the training data.

**load\_models(self, date)**

* **Parameters**:
  + date (str): Date string used to load specific model versions.
* **Return Type**: int
* **Description**: Loads the trained models from the file system.

**print\_model(self, name, model)**

* **Parameters**:
  + name (str): Name of the model.
  + model (Pipeline): Model pipeline.
* **Return Type**: None
* **Description**: Prints the parameters and selected features of the model.

**preprocess(self, text)**

* **Parameters**:
  + text (str): Input text.
* **Return Type**: sparse matrix
* **Description**: Transforms the input text using the TF-IDF vectorizer.

**predict\_pdf(self, url)**

* **Parameters**:
  + url (str): URL of the PDF.
* **Return Type**: list of str
* **Description**: Extracts text from the PDF at the given URL and makes predictions.

**predict\_Files(self, df)**

* **Parameters**:
  + df (pd.DataFrame): DataFrame containing URLs of PDFs.
* **Return Type**: pd.DataFrame
* **Description**: Extracts text from the PDFs, makes predictions, and returns the results.

**predict(self, text\_list)**

* **Parameters**:
  + text\_list (list of str): List of input texts.
* **Return Type**: dict
* **Description**: Makes predictions using the loaded models for each input text.

**expand\_Models(self)**

* **Parameters**: None
* **Return Type**: RandomForestClassifier
* **Description**: Loads the models and returns the RandomForestClassifier with best parameters from GridSearchCV.

**Usage**

**Preprocessing Text**

To preprocess text using the TextPreprocessor class:

python

Copy code

preprocessor = TextPreprocessor(n\_jobs=4)

preprocessed\_text = preprocessor.transform(['Sample text to preprocess.'])

print(preprocessed\_text)

**Training Models**

To train models using the TrainingModels class:

python

Copy code

trainer = TrainingModels()

trainer.Load\_data()

trainer.train()

**Predicting Classes**

To predict the class of a new text or PDF document:

python

Copy code

predictions = trainer.predict(['Sample text to classify.'])

print(predictions)

pdf\_url = 'http://example.com/sample.pdf'

pdf\_prediction = trainer.predict\_pdf(pdf\_url)

print(pdf\_prediction)

# FastAPI Application and Routes

This FastAPI application provides endpoints for making predictions based on URLs and uploaded files.

**URLRequest Class**

* **Parameters**:
  + url (str): The URL of the PDF to be processed.
* **Return Type**: Not applicable (it's a data model class).
* **Description**: This Pydantic model defines the request body structure for the /predict/ endpoint.

**CSVRequest Class**

* **Parameters**:
  + csv\_path (str): The path to the CSV file.
* **Return Type**: Not applicable (it's a data model class).
* **Description**: This Pydantic model defines the request body structure for handling CSV file paths.

**predict Function**

* **Endpoint**: /predict/
* **HTTP Method**: POST
* **Parameters**:
  + request (URLRequest): Request body containing the URL of the PDF.
* **Return Type**: dict
* **Description**:
  + Receives a URL in the request body.
  + Uses the model\_instance to predict based on the PDF URL.
  + Returns a JSON response with the prediction and confidence.

**predict\_file Function**

* **Endpoint**: /predict\_file/
* **HTTP Method**: POST
* **Parameters**:
  + file (UploadFile): The uploaded file.
* **Return Type**: FileResponse
* **Description**:
  + Handles file uploads (CSV, XLS, XLSX).
  + Saves the uploaded file to a temporary location.
  + Reads the file into a DataFrame.
  + Uses model\_instance to predict based on the DataFrame.
  + Returns the prediction results as a CSV file.

**predict\_url\_file Function**

* **Endpoint**: / predict\_url\_file/
* **HTTP Method**: POST
* **Parameters**:
  + file (UploadFile): The uploaded file.
  + Number (Form): Number of lines extracted at once
* **Return Type**: FileResponse
* **Description**:
  + Handles file uploads (CSV, XLS, XLSX).
  + Saves the uploaded file to a temporary location.
  + Reads the file into a DataFrame.
  + Sample the data by Number.
  + Uses model\_instance to predict sample based on the DataFrame.
  + Returns the sample result as a table in the api

**extract\_text\_from\_pdf Function**

* **Endpoint**: / extract\_text\_from\_pdf/
* **HTTP Method**: POST
* **Parameters**:
  + request (URLRequest): the URL link for the pdf
* **Return Type**: Dict
* **Description**:
  + Extract text from a pdf
  + Using OCR if the pdf doesn’t have text

**root Function**

* **Endpoint**: /
* **HTTP Method**: GET
* **Parameters**: None
* **Return Type**: FileResponse
* **Description**:
  + Serves the index.html file from the templates directory.

**TrainingModels Initialization**

**model\_instance**

* **Description**:
  + An instance of the TrainingModels class.
  + Loads the models using model\_instance.load\_models(date="").

**FastAPI Application Setup**

* **Description**:
  + The FastAPI application is instantiated with app = FastAPI().
  + The APIRouter is instantiated and used to define routes.
  + The static files directory is mounted to serve static files.
  + The router is included in the main app using app.include\_router(router).