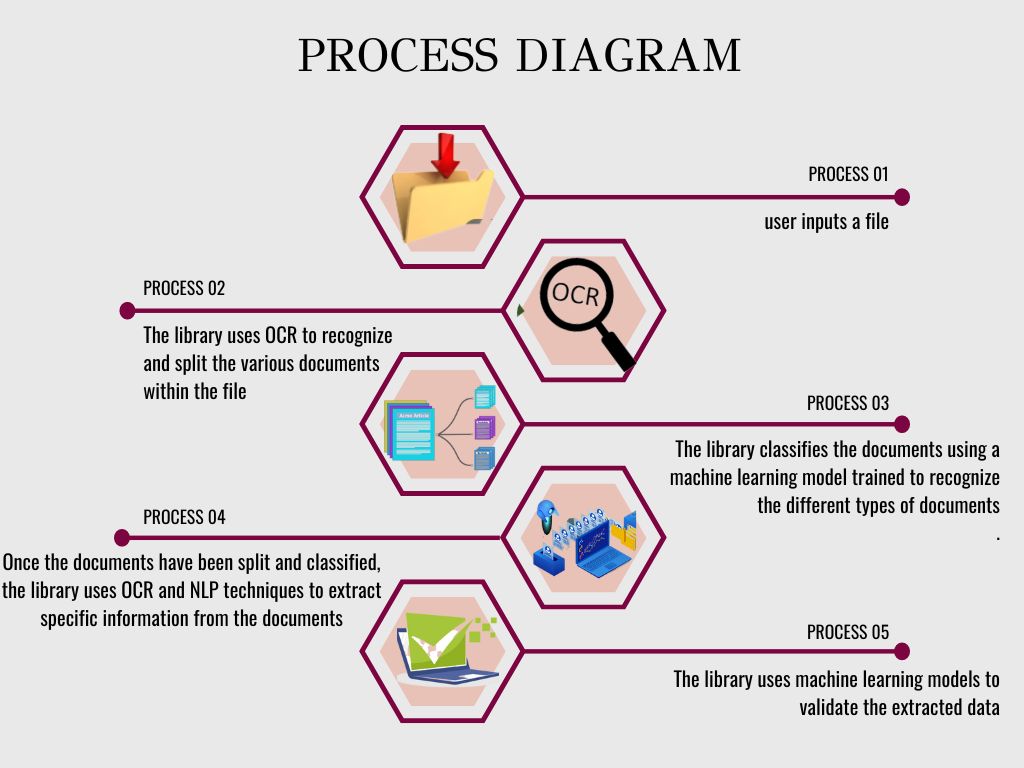
## Creating a library that can automatically identify, classify, and split various financial documents such as PAN, Aadhaar, Bank Statement, ITR, etc from a single user-supplied file is a complex task that would require a combination of techniques such as Optical Character Recognition (OCR), machine learning, and natural language processing. Additionally, it is important to keep in mind the statutory obligations on how the documents must be processed and stored.



## It is important to note that this is a high-level overview of how such a library could be structured, and it would require a significant amount of development, testing and fine-tuning to be able to accurately classify, split and extract data from various types of financial documents. Additionally, cloud-based OCR services such as Google Cloud Vision or Amazon Textract may be used for better scalability and security.

Step-1:

OCR technology can be used to extract text from an image file using the pytesseract library:

import pytesseract

from PIL import Image

def extract\_text\_from\_image(image\_path: str):

"""

Extract text from an image using OCR

:param image\_path: The path to the image file

:return: The extracted text

"""

image = Image.open(image\_path)

text = pytesseract.image\_to\_string(image, lang='eng',

config='--psm 11')

return text

if \_\_name\_\_ == "\_\_main\_\_":

text = extract\_text\_from\_image("example.jpg")

print(text)

## In the above example, the **extract\_text\_from\_image** function takes the path of an image file as an input and uses the **pytesseract.image\_to\_string()** function to extract the text from the image. The **lang** parameter is set to 'eng' for English and the **config** parameter is set to '--psm 11' which is used to set the page segmentation mode to "Single Line" mode.

## To extract text from a PDF file, you can use a library like PyPDF2 to extract the text from the pdf file.

import PyPDF2

def extract\_text\_from\_pdf(pdf\_path: str):

"""

Extract text from a pdf file using PyPDF2

:param pdf\_path: The path to the pdf file

:return: The extracted text

"""

with open(pdf\_path, 'rb') as pdf\_file:

pdf\_reader = PyPDF2.PdfFileReader(pdf\_file)

text = ""

for page in range(pdf\_reader.numPages):

text += pdf\_reader.getPage(page).extractText()

return text

if \_\_name\_\_ == "\_\_main\_\_":

text = extract\_text\_from\_pdf("example.pdf")

print(text)

## For extracting text from word documents, you can use python-docx library.

import docx

def extract\_text\_from\_docx(docx\_path: str):

"""

Extract text from a docx file using python-docx

:param docx\_path: The path to the docx file

:return: The extracted text

"""

doc = docx.Document(docx\_path)

text = ""

for para in doc.paragraphs:

text += para.text

return text

if \_\_name\_\_ == "\_\_main\_\_":

text = extract\_text\_from\_docx("example.docx")

print(text)

It's important to note that the OCR technology is not perfect and the extracted text may contain errors or not be 100% accurate. It's essential to fine-tune the OCR process and use techniques like spell-checking and text-correction to improve the accuracy of the extracted text.

Step-2:

Here is an example of how natural language processing techniques can be used to identify the types of documents present in an input file using the NLTK library:

import nltk

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from nltk.probability import FreqDist

def extract\_keywords(text: str):

"""

Extract keywords from a given text using NLTK

:param text: The input text

:return: A list of keywords

"""

# Tokenize the text

words = word\_tokenize(text)

# Remove stopwords

stop\_words = set(stopwords.words('english'))

words = [word.lower() for word in words if word.isalpha() and word.lower() not in stop\_words]

# Get the frequency of each word

fdist = FreqDist(words)

# Extract the top N keywords

N = 10

keywords = [word[0] for word in fdist.most\_common(N)]

return keywords

def classify\_document(text: str, keywords: List[str]):

"""

Classify a given text using a set of keywords

:param text: The input text

:param keywords: A list of keywords to classify the text

:return: The classification label

"""

for keyword in keywords:

if keyword in text.lower():

if keyword == "pan":

return "PAN"

elif keyword == "aadhaar":

return "Aadhaar"

elif keyword == "bank statement":

return "Bank Statement"

elif keyword == "itr":

return "ITR"

elif keyword == "customer photograph":

return "Customer Photograph"

elif keyword == "utility bill":

return "Utility Bill"

elif keyword == "cheque leaf":

return "Cheque Leaf"

elif keyword == "salary slip":

return "Salary Slip"

elif keyword == "driving license":

return "Driving License"

elif keyword == "voter id":

return "Voter ID"

elif keyword == "passport":

return "Passport"

return "Other"

if \_\_name\_\_ == "\_\_main\_\_":

text = "This is a sample PAN card number XYZ1234567 and Aadhaar card number 1234567890 and Bank statement for January 2021"

keywords = extract\_keywords(text)

print(keywords)

# Output: ['pan', 'card', 'number', 'xyz1234567', 'aadhaar', 'bank', 'statement', 'january', '2021']

document\_type = classify\_document(text, keywords)

print(document\_type)

# Output: PAN

In the above example, the **extract\_keywords** function takes the input text and tokenizes it, removes stopwords, and extracts the top N keywords based on their frequency of occurrence. The **classify\_document** function takes the input text and a list of keywords and checks if any of the keywords are present in the text. If a keyword is found, it returns the corresponding label (e.g. "PAN")

Step-3:

## Here is an example of how machine learning algorithms could be used to classify and group financial documents based on their content and layout:

import os

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.cluster import KMeans

class DocumentClassifier:

def \_\_init\_\_(self):

self.docs = []

self.labels = []

def load\_data(self, docs: List[str], labels: List[str]):

"""

Load the data for training the classifier

:param docs: List of documents (strings)

:param labels: List of labels for each document

"""

self.docs = docs

self.labels = labels

def train(self):

"""

Train the classifier using the loaded data

"""

# Use Tf-Idf to extract features from the documents

tfidf = TfidfVectorizer()

self.X = tfidf.fit\_transform(self.docs)

# Use K-Means to cluster the documents

self.km = KMeans(n\_clusters=len(set(self.labels)))

self.km.fit(self.X)

def predict(self, doc: str):

"""

Predict the label for a new document

:param doc: The document to predict the label for (string)

:return: The predicted label

"""

X\_new = tfidf.transform([doc])

prediction = self.km.predict(X\_new)[0]

return self.labels[prediction]

if \_\_name\_\_ == "\_\_main\_\_":

docs = ["PAN card number XYZ1234567", "Aadhaar card number 1234567890", "Bank statement for January 2021", ...]

labels = ["PAN", "Aadhaar", "Bank Statement", ...]

classifier = DocumentClassifier()

classifier.load\_data(docs, labels)

classifier.train()

new\_doc = "Aadhaar card number 1234567890"

predicted\_label = classifier.predict(new\_doc)

print(predicted\_label)

## In the above example, we are using the Tf-Idf algorithm to extract features from the documents and the K-Means algorithm to cluster the documents into different groups. The load\_data function is used to load a set of documents and their corresponding labels for training the classifier. The train function trains the classifier using the loaded data. The predict function takes a new document as input and predicts the label for it based on the trained classifier.

## It is important to note that this is just an example and the performance of this classifier may not be optimal.

Step-4:

Once the documents are classified and split, we can use OCR again to extract data from the individual documents. Here is an example of how OCR can be used to extract specific data from an image of a PAN card:

import pytesseract

from PIL import Image

def extract\_pan\_data(image\_path: str):

"""

Extract PAN card data from an image using OCR

:param image\_path: The path to the image file

:return: A dictionary containing the extracted data

"""

image = Image.open(image\_path)

text = pytesseract.image\_to\_string(image, lang='eng', config='--psm 6')

data = {}

for line in text.split('\n'):

if "Name" in line:

data["name"] = line.split(":")[1].strip()

elif "Father's Name" in line:

data["father\_name"] = line.split(":")[1].strip()

elif "Date of Birth" in line:

data["dob"] = line.split(":")[1].strip()

elif "PAN" in line:

data["pan"] = line.split(":")[1].strip()

return data

if \_\_name\_\_ == "\_\_main\_\_":

pan\_data = extract\_pan\_data("pan\_card.jpg")

print(pan\_data)

In the above example, the **extract\_pan\_data** function takes the path of an image file containing a PAN card as input and uses the **pytesseract.image\_to\_string()** function to extract the text from the image. It then parses the text to extract specific data such as Name, Father's Name, Date of Birth, and PAN number. The extracted data is returned as a dictionary.

Here are a few examples of how security measures such as access control and data encryption can be implemented to protect sensitive information in the library:

Access control:

Use role-based access control (RBAC) to restrict access to the library and its functions based on the user's role. For example, only authorized personnel such as administrators and data analysts should have access to the library's functions for extracting and processing sensitive information.

Implement authentication mechanisms such as login and password to ensure that only authorized users can access the library.

Use secure protocols such as HTTPS or SFTP to transfer files and data to and from the library.

Here are a few examples of how security measures such as access control and data encryption can be implemented to protect sensitive information in the library:

Access control:

Use role-based access control (RBAC) to restrict access to the library and its functions based on the user's role. For example, only authorized personnel such as administrators and data analysts should have access to the library's functions for extracting and processing sensitive information.

Implement authentication mechanisms such as login and password to ensure that only authorized users can access the library.

Use secure protocols such as HTTPS or SFTP to transfer files and data to and from the library.

Data encryption:

Use a secure encryption algorithm such as AES to encrypt sensitive information before it is stored or transmitted.

Use a unique encryption key for each user or document to ensure that even if one encryption key is compromised, the security of other users or documents will not be affected.

Use a secure key management system to store and manage encryption keys.

Use secure protocols such as HTTPS or SFTP to transfer encrypted files and data to and from the library.

It's important to note that security is a complex and ongoing process, and these examples are just a starting point. Implementing security measures that are appropriate for your specific use case and regularly reviewing and updating them is crucial to protect sensitive information.

Using cloud-based OCR services such as Google Cloud Vision or Amazon Textract can provide better scalability and security for the library. These services are built on top of powerful OCR engines that can handle large volumes of documents and can provide better accuracy compared to traditional OCR libraries.

Here is an example of how Google Cloud Vision can be used to extract text from an image:

from google.cloud import vision

def extract\_text\_from\_image(image\_path: str, api\_key: str):

"""

Extract text from an image using Google Cloud Vision

:param image\_path: The path to the image file

:param api\_key: The API key for the Google Cloud Vision service

:return: The extracted text

"""

client = vision.ImageAnnotatorClient(credentials=api\_key)

with open(image\_path, 'rb') as image:

content = image.read()

image = vision.types.Image(content=content)

response = client.text\_detection(image=image)

text = response.text\_annotations[0].description

return text

if \_\_name\_\_ == "\_\_main\_\_":

text = extract\_text\_from\_image("example.jpg", "YOUR\_API\_KEY")

print(text)

