```
# installing important libraries
!pip install pytesseract
!pip install pdf2image
!sudo apt install tesseract-ocr
!apt-get install -y poppler-utils
!pip install imbalanced-learn
```

```
Requirement already satisfied: pytesseract in /usr/local/lib/python3.10/dist-packages (0.3.10)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from pytesseract) (23.2)
Requirement already satisfied: Pillow>=8.0.0 in /usr/local/lib/python3.10/dist-packages (from pytesseract) (9.4.0)
Requirement already satisfied: pdf2image in /usr/local/lib/python3.10/dist-packages (1.16.3)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from pdf2image) (9.4.0)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
tesseract-ocr is already the newest version (4.1.1-2.1build1).
0 upgraded, 0 newly installed, 0 to remove and 18 not upgraded.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
poppler-utils is already the newest version (22.02.0-2ubuntu0.2).
0 upgraded, 0 newly installed, 0 to remove and 18 not upgraded.
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (0.10.1)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.11.3)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.2.0)
```

```
# importing the library
import tracemalloc
import os
import platform
from tempfile import TemporaryDirectory
from pathlib import Path
import pytesseract
from pdf2image import convert_from_path
from PIL import Image
import cv2
import pytesseract
import os
import numpy as np
import pandas as pd
import re
from pdf2image import convert_from_bytes
import re,time
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
nltk.download('stopwords')
from nltk.stem.snowball import SnowballStemmer
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.linear_model import SGDClassifier
import spacy
from sklearn import preprocessing
import pickle
#library that contains punctuation
import string

#loading the english language small model of spacy
en = spacy.load('en_core_web_sm')
sw_spacy = en.Defaults.stop_words
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
# Defining function to use tracemalloc
def tracing_start():
    tracemalloc.stop()
    print("nTracing Status : ", tracemalloc.is_tracing())
    tracemalloc.start()
    print("Tracing Status : ", tracemalloc.is_tracing())
def tracing_mem():
    first_size, first_peak = tracemalloc.get_traced_memory()
```

```
peak = first_peak/(1024*1024)
print("Peak Size in MB - ", peak)
```

We will convert the pdf data to a label dataframe with every 1st page with form type so that this problem will be easy to solve

```
# function to create supervised data
def supervised_df_creation():
    # iterating over the folder to get the all the files
    for file in os.listdir('/content/'):
        # check if file ends with extension pdf then only continue with process
        # this is a fail safe if program fails while processing the file because we can start again from last save point
        if file.endswith('.pdf'):
            # check if supervised data file exists
            if os.path.exists('/content/file.csv'):
                # read the file if file exist
                df_data= pd.read_csv('file.csv')
                # check if file name is the supervised file and if present then continue to next file
                if file in list(df_data['file']):
                    continue
            print(file)

            # return the pdf pages as image format so that we can convert it into text afterwards
            #it Read in the PDF file at 500 DPI
            pdf_pages = convert_from_path(file, 500)

            # Iterate through all the pages stored above
            for page_enumeration, page in enumerate(pdf_pages, start=1):

                try:
                    # intialize a list to get the image file
                    image_file_list = []
                    # create a dataframe for storing the type and first page as text
                    pages_df = pd.DataFrame(columns=['file','text', 'type'])

                    # we will only check first page because on 1st page only we will get the form type so no need of reading ,preprocessing
                    if page_enumeration == 1:
                        # save the pages so that we can refer it later
                        with TemporaryDirectory() as tempdir:
                            filename = f"{tempdir}\page_{page_enumeration:03}.jpg"
                            page.save(filename, "JPEG")
                            image_file_list.append(filename)
                        # open the text file
                        # with open(text_file, "a") as output_file:
                            # Open the file in append mode so that

                        # Iterate from 1 to total number of pages
                        for image_file in image_file_list:
                            text = str(((pytesseract.image_to_string(Image.open(image_file)))))

                        # we get  the form type from the data using regex matching
                        # we do this to make the problem into supervised learning
                        # if form is present we can extract the name righly else make it other
                        if re.search('FORM\s\w+',text):
                            form_type = re.search('FORM\s\w+',text)[0]
                        elif re.search('Form\s\w+',text):
                            form_type = re.search('Form\s\w+',text)[0]
                        else:
                            form_type = 'other'

                        # after getting the form type we will update the dataframe
                        pages_df = pages_df.append({'file': file,'text': text, 'type': form_type}, ignore_index=True)

                        # if file exists then we will append the df else we will create new file
                        # we do this becasue the data is large and it might crash the system so we can start again but it will start from l
                        if os.path.exists('/content/file.csv'):
                            pages_df.to_csv('file.csv',mode='a',header=False)
                        else:
                            pages_df.to_csv('file.csv',mode='w')

                        # del the variables
                        del filename,page,image_file_list,pdf_pages,text
                except Exception as err:
                    print(err)
```

```
  # function to clean text removing new line character , extra spaces, remove punctuation and number
  def clean_txt(text):
      text = re.sub("'"," ",text)
      text = re.sub("(\\W)+"," ",text)
      text = re.sub("\n"," ",text)
      text = re.sub("(\\d)+"," ",text)
      punctuationfree="".join([i for i in text if i not in string.punctuation])

      return punctuationfree


  # function to make word from character\
  def join_text(text):
      text = " ".join([i for i in text])
      return text


# calling function to create supervised df and saving into file so that we can use it anytime we want to use and file size is only 220kb
tracing_start()
start = time.time()
supervised_df_creation()
```

```
    nTracing Status :  False
    Tracing Status :  True
```

```
# read the file and resey the index
df = pd.read_csv('/content/file.csv')
df = df.reset_index(drop=True)
print(df.head())
```

```
       Unnamed: 0          file  \
    0           0  01057890.pdf
    1           0  01057240.pdf
    2           0  01074669.pdf
    3           0  00104263.pdf
    4           0  00104564.pdf

                                                 text      type
    0  41/15/2001 09:51 FAX 2123065325 LISTING QUALIE...    Form 8
    1  a 2QIF-1|$92-5 |\n\nFORM D UNITED STATES |\n\n...    FORM D
    2   \n\nare, CONFIDENT IAL\n\n- UNITED STATES\nSE...  Form 13F
    3   \n\nras\n\nVow sryuyY\n\n    \n \n\nOMB\n\nOM...  FORM 13F
    4  ‘t3rbuN ny pL were:\n\n2114] /o1 FEB 5 2 be\n\...  Form 13F
```

preprocessing the type column

```
# preprocessing the type column

# replacing wrong form type and making it correcct
replace_values = {'Form ONITED': 'other', 'FORM LIMITED': 'other', 'FORM L3F': 'Form 13F', 'FORM TA': 'other','FORM X': 'FORM X-17','Form 1i
df = df.replace({"type": replace_values})

# making the type lower so that multiple Form 13F & FORM 13F can be same
df['type'] = df['type'].str.lower()

# encoding the type for better processing
label_encoder = preprocessing.LabelEncoder()
df['type']= label_encoder.fit_transform(df['type'])



# preprocessig the dataframe
# making all the text lower
df['text'] = df['text'].str.lower()

# remove all the unnecessary words
df['text'] = df.text.apply(clean_txt)

# we will stem the word
stemmer = SnowballStemmer("english")

# spliting text on blank space
df['text'] = df['text'].str.split()
```

```
# applying the stemmer on data
df['text'] = df['text'].apply(lambda x: [stemmer.stem(y) for y in x]) # Stem every word.

# removing stop word
for item in list(df['text']):
    ietm_copy = item.copy()
    for i in ietm_copy:
        if i in sw_spacy:
            item.remove(i)

# join the data
df['text'] = df.text.apply(join text)
```

vectorize and train the model

```
# get the tfidf vectorizer so we can transform the data for machine learning
# we use l2 regularization because it give more accuracy
vectorizer = TfidfVectorizer(stop_words='english',norm='l2')
tfidf = vectorizer.fit_transform(df['text'])

# getting x and y for ML modelling
X = pd.DataFrame(tfidf.toarray())
y = df['type']

# using the train test split function
X_train, X_test, y_train, y_test = train_test_split(X,y , random_state=10,  test_size=0.25,  shuffle=True)


model = SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3, random_state=42)
# fit the model
model.fit(X_train, y_train)


# saving the pickle file for future purpose
with open('model_pkl', 'wb') as files:
    pickle.dump(model, files)

y_pred = model.predict(X_test)
print((metrics.accuracy_score(y_test, y_pred)*100))
print(classification_report(y_test,y_pred))
```

```
92.3076923076923
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         0
           2       0.97      0.97      0.97        40
           3       1.00      1.00      1.00         1
           4       0.82      0.90      0.86        10
           5       1.00      1.00      1.00         5
           6       0.86      0.67      0.75         9

    accuracy                           0.92        65
   macro avg       0.78      0.76      0.76        65
weighted avg       0.94      0.92      0.93        65

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defi
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defi
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defi
  _warn_prf(average, modifier, msg_start, len(result))
```

```
end = time.time()
print("time elapsed {} milli seconds".format((end-start)*1000))
tracing_mem()
```

```
time elapsed 21303.054332733154 milli seconds
Peak Size in MB -  21.30649185180664
```