```
#import the necessary modules
import csv
import akkadian
import akkadian.transliterate as akk
import sys
import os
from transformers import pipeline
#create the lists which will be used in  removing the duplicates of the dataset
temp_list_cun = []
temp_list_translit = []
#create two command-line arguments - the first mentioning how many lines
#of the dataset will be translated, and second argument mentions the file
#in which to store the translated lines
argv1 = int(sys.argv[1])
argv2 = sys.argv[2]
#establish the number of lines of 1000, which will be translated
#when running the code in command line
num_lines = 1000
#check if the path to the file to store lines exists, if yes remove it and
#create it in the code by opening the file - in binary mode, with the possiblity
#to append it - because some characters
#of the output are not accepted by a text file
if os.path.exists(argv2):
        os.remove(argv2)
f = open(argv2, "ab")
#import the language model which will translate the cuneiform from HuggingFace
#also create the pipeline to use in the model
pipe = pipeline("text2text-generation", model="praeclarum/cuneiform")
#open the csv file in which the cuneiform lines are stored
#also specify the encoding
with open(r"C:\\Users\keric\train.csv", "r", encoding="utf-8") as csvfile:
#read it
        akkad = csv.reader(csvfile)
        #create a counter to number the read lines
        counter = 0
        #create a counter to number the translated lines
        processed_counter = 0
        #loop through the read lines
        for row in akkad:
        #make sure any error is caught in try-except block
                try:
                #jump over first read line, which contains no cuneiform lines
                #and go to the next
                        if counter == 0:
                                counter +=1
                                continue
                                #if the number of read lines is smaller than the number
                                #of lines of the first argument of the command line, go to the next line
                        if counter < argv1:
                                counter +=1
                                continue
                                #if the number of read lines is larger than the number
                                #of lines of the first argument of the command line to which
                                #1000 lines to be translated on the command line are added and one subtracted, exit
                        if counter > (argv1 + num_lines - 1):
                                break
                        #extract the first string in each of the lists containing a cuneiform line
                        cuneiform = row[0]
                        #remove duplicates, by storing the cuneiform lines into a list and
                        #refusing to add same lines to the list
                        if cuneiform not in temp_list_cun:
                                temp_list_cun.append(cuneiform)
                                #transliterate them with the akkadian package Hidden Markov Model function
                                transliterated = akk.transliterate_hmm(cuneiform)
                                #translate the transliterated, using the model with the pipeline
                                translated = pipe(transliterated)
                                temp_list_translit.append(transliterated)
                                #write the number of each cuneiform line into the file
                                f.write("\nCuneiform line number:".encode())
                                counter_str = str(counter)
                                f.write(counter_str.encode())
                                f.write("\n--------------------------------------------------\n".encode())
                                #write the cuneiform line to the file
                                f.write(cuneiform.encode())
                                #write its transliteration to the file
                                f.write("\nTransliterated:\n".encode())
                                f.write("--------------------------------------------------\n".encode())
                                f.write(transliterated.encode())
                                #write the transliterated's translation to the file
                                #removing the "generated text" section and keeping only the
                                #translation itself
                                f.write("\nTranslated:\n".encode())
                                f.write("--------------------------------------------------\n".encode())
                                f.write(translated[0]["generated_text"].encode())
                                f.write("\n\n\n".encode())
                        #increase the counter of the processed lines
```

```
                        processed_counter +=1
                        #increase the counter numbering the translated lines
                        counter +=1
                        #write a message on the command line if the number of processed lines
                        #is divided by 100
                        if processed_counter%100 == 0 :
                                print(processed_counter, " lines are translated")
                except:
                        #close the try-except block and print an error message if any of these
                        #operations above goes wrong
                                print("Error!")
#close the csv file and the binary file
csvfile.close()
f.close()

#I had run this code on 4 command lines at the same time as follows:
#line 1 - the first argument was 0, because I started from 0 lines and translated 999
#line 2 - the first argument was 1000, because I started from 1000 lines and translated 999
#line 3 - the first argument was 2000, because I started from 2000 lines and translated 999
#line 4 - the first argument was 3000, because I started from 3000 lines and translated 999
#on each line I stored the translated lines into a separate file
#which I then assembled manually into a single file

#I wrote my script in VisualStudio and used my laptop to run it
#it seems that you need a special font to see the cuneiform lines in the output file
#and I downloaded and attached as a file the CuneiformComposite file, which
#needs to be also downloaded and installed on your computer and then
#the output file can be opened with Notepad

#here is the link to download the cuneiform font, if you want to download the
#CuneiformComposite file from a website
#https://oracc.museum.upenn.edu/doc/help/visitingoracc/fonts/
```