ISW preprocessing

Install packages

```
%%bash
# run pip upgrade if 'KeyboardInterrupt' error occurs
# pip install --upgrade pip

pip install nltk num2words
pip install -U scikit-learn
```

Import and download all dependecies

```
import pandas as pd
import numpy as np
import nltk
import string
import pickle
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re
from num2words import num2words
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
import pandas as pd
import string
from zipfile import ZipFile, ZipInfo
from pathlib import Path
import os

nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

Define preprocessing functions

```
# Functions
def to_lower_case(text):
  return "".join([i.lower() for i in text])

stop_punctuation = string.punctuation
def remove_punctuation(text):
  return "".join([i for i in text if i not in stop_punctuation])

def remove_long_dash(text):
  return re.sub(r'—', ' ', text)

def remove_urls(text):
  return re.sub(r'http\S+', '', text)

def remove_one_letter_words(tokens):
  return list(filter(lambda token: len(token) > 1, tokens))

def tokenize_text(text):
  return nltk.tokenize.word_tokenize(text)

stop_words = set(nltk.corpus.stopwords.words('english'))
avoid_stop_words = set(["not","n't","no"])
stop_words = stop_words - avoid_stop_words

def remove_stop_words(tokens):
  return [i for i in tokens if i not in stop_words]

def do_stemming(tokens):
  ps = nltk.PorterStemmer()
  return [ps.stem(word) for word in tokens]

def do_lemmatization(tokens):
  wn = nltk.WordNetLemmatizer()
  return [wn.lemmatize(word) for word in tokens]

def remove_numeric_words(text):
  return re.sub(r'\S*\d+\S*', '', text)

def convert_nums_to_words(data):
  tokens = data
  new_text = []
  for word in tokens:
    if word.isdigit():
      if int(word)<1000000000:
        word = num2words(word)
      else:
        word = ""
    new_text.extend(tokenize_text(re.sub("(-|,\s?)|\s+", " ", word)))
  return new_text

def do_preprocessing(data):
  text_clean = data
  text_clean = remove_urls(text_clean)
  text_clean = remove_punctuation(text_clean)
  text_clean = remove_long_dash(text_clean)
  text_clean = to_lower_case(text_clean)
```

Zip opening

```python
# specifying the zip file name
file_name = "/work/isw_scrapping_res.zip"

df = pd.DataFrame(columns = ["Name", "Date", "Text"])
print("Openning zip in read mode")
# opening the zip file in READ mode
with ZipFile(file_name, 'r') as zipfile:
      for file in zipfile.infolist():
        if not ZipInfo.is_dir(file):
          filename = file.filename.rsplit('/', 1)[1].split('.')[0]
          date = filename.replace("assessment-", "")
          text = zipfile.read(file.filename).decode('utf-8')
          df = df.append({"Name": filename, "Date": date, "Text": text}, ignore_index = Tr
```

```
Openning zip in read mode
```

TF-IDF creation

```python
print("Find tokens")
df["Tokens"] = df["Text"].apply(lambda d: " ".join(do_preprocessing(d)))

filenames = df["Name"]
dates = df["Date"]

print("Create vectors")
tfidf = TfidfVectorizer(smooth_idf=True,use_idf=True)
vectors = tfidf.fit_transform(df["Tokens"])

# store content
with open("/work/results/tfidf.pkl", "wb") as handle:
  pickle.dump(tfidf, handle)

feature_names = tfidf.get_feature_names_out()
dense = vectors.todense()
denselist = dense.tolist()
df = pd.DataFrame(denselist, columns=feature_names)
dictionaries = df.to_dict(orient='records')

print("Into result")
res = __builtins__.zip(filenames, dates, dictionaries)
res_df = pd.DataFrame(res, columns=["Name","Date","Keywords"])
res_df["Keywords"] = res_df["Keywords"].apply(lambda d: {k: v for k, v in d.items() if v >
res_df
```

```
Find tokens
Create vectors
Into result
```

	Name object	Date object	Keywords object
	assessment-... 0.3%	2023-03-01 .... 0.3%	{'ability': 0.01... 0.3%
	assessment-... 0.3%	2023-03-02 ...... 0.3%	{'accept': 0.0... 0.3%
	391 others 99.5%	391 others 99.5%	391 others 99.5%

0	assessment-2023-03-01	2023-03-01	{'ability': 0.012740828616...	
1	assessment-2023-03-02	2023-03-02	{'accept': 0.0091109105166...	
2	assessment-2023-03-03	2023-03-03	{'able': 0.008175006201...	
3	assessment-2023-03-04	2023-03-04	{'abandon': 0.0266161146270...	
4	assessment-2023-03-05	2023-03-05	{'abbreviated': 0.017019927623...	
5	assessment-2023-03-06	2023-03-06	{'abandoned': 0.008803591461...	
6	assessment-2023-03-07	2023-03-07	{'ability': 0.006542105673...	
7	assessment-2023-03-08	2023-03-08	{'able': 0.006142418779...	
8	assessment-2023-03-09	2023-03-09	{'accepting': 0.012891000986...	
9	assessment-2023-03-10	2023-03-10	{'abducted': 0.022872555268...	

Forming zip with .csv output

```
filename = "tfidf-result"
compression_options = dict(method='zip', archive_name=f'{filename}.csv')
res_df.to_csv(f'/work/results/{filename}.zip', compression=compression_options, index=Fals
```