main.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

#####################################################################
# title:    BBS for Independence - data analysis using NLP
# subtitle: HSA in the ABC of Computational Text Analysis
# author:   Josias Bruderer, Universität Luzern
# date:     30. August 2021
# desc:     this script manages the whole analisis of textfiles.com
#####################################################################

# Preparations [R1]
"""
The following lines of code is used for preparing our environment.
"""

# load the necessary libraries
import os
import shutil
import sys
import time
from pathlib import Path
import textacy
import spacy
import pandas as pd
import scattertext as st
from plotnine import *
import numpy as np
from multiprocessing import Pool
from multiprocessing.managers import BaseManager
import string
from octis.preprocessing.preprocessing import Preprocessing
# from octis.dataset.dataset import Dataset
# from octis.evaluation_metrics.diversity_metrics import TopicDiversity
# from octis.models.LDA import LDA
# from wordcloud import WordCloud
import csv

project_path = Path.cwd()

# prepare to load project specific libraries
if project_path not in sys.path:
    sys.path.append(str(project_path))

# import modules
from modules import wrangler
from modules import helpers
from modules import nlp_pool
from modules import years

# Data Wrangling [R2]
"""
In this section the required data is downloaded and preprocessed (f.E. unzipped).
The module data_wrangler will be used for this.
"""

number_of_threads = 24

"""
skip_steps = [] # skip nothing
skip_steps = ["download", "cleaning"] # use this after modification on metadata_file_filter
skip_steps = ["download", "cleaning", "metadata-filtering", "modeling", "analysis_freq", "analysis_advance_preparation",
              "analysis_scattertext", "analysis_year", "analysis_octis", "analysis_entities"] # the full list
"""
skip_steps = [] # the full list

data_url = "http://archives.textfiles.com/[name].zip"
data_names = ["100", "adventure", "anarchy", "apple", "art", "artifacts", "bbs", "computers", "conspiracy", "digest",
              "drugs", "etext", "exhibits", "floppies", "food", "fun", "games", "groups", "hacking", "hamradio",
              "history", "holiday", "humor", "internet", "law", "magazines", "media", "messages", "music", "news",
              "occult", "phreak", "piracy", "politics", "programming", "reports", "rpg", "science", "sex", "sf",
              "stories", "survival", "tap", "ufo", "uploads", "virus",
              "fidonet-on-the-internet"]  # categories to download
data_names_exclude = ["fidonet-on-the-internet", "tap", "floppies", "exhibits", "artifacts",
                      "piracy", "art", "magazines", "digest"]  # categories that are excluded and removed from data_names
file_filter = "^.*(\.(jpe?g|png|gif|bmp|zip|mp3|wav))|index\.html?$"  # use this to exclude by filenames
metadata_file_filter = "(x['metadata']['charratioB'] > 0.95) & " \
                       "(x['metadata']['length'] > 300) & (x['metadata']['length'] < 30000)"
metadata_file_filter_declaration = "(x['metadata']['charratioB'] > 0.8) & " \
                                   "(x['metadata']['length'] > 300) & (x['metadata']['length'] < 30000)"
data_dir = Path(project_path / "02_datasets/")
models_dir = Path(project_path / "03_workspace/models/")
analysis_dir = Path(project_path / "03_workspace/analysis/")
octis_dataset_dir = Path(project_path / "03_workspace/OCTIS/preprocessed_datasets/")
tmp_dir = Path(project_path / ".tmp/")

data_dir.mkdir(parents=True, exist_ok=True)
models_dir.mkdir(parents=True, exist_ok=True)
tmp_dir.mkdir(parents=True, exist_ok=True)

threads = []

if "download" not in skip_steps:
    # prepare the threads for loading data
    for names in helpers.chunker_list(data_names, number_of_threads):
        threads.append(wrangler.loader(tmp_dir, data_dir, data_url, names))

    # start all threads
    for thread in threads:
        thread.start()

    # wait for all threads to finish their work
    running = True
    while running:
        running = False
        for thread in threads:
            if thread.is_alive():
                running = True
                print("waiting for threads to finish...")
                time.sleep(1)

    print("data downloaded successfully")

if "cleaning" not in skip_steps:
    # prepare the threads for cleaning up stuff
    threads = []
    dataset = {}

    # prepare the threads
    for names in helpers.chunker_list(data_names, number_of_threads):
        threads.append(wrangler.cleaner(data_dir, names, file_filter))

    # add declaration
    threads.append(wrangler.cleaner(data_dir, ["declaration"], file_filter))

    # start all threads
    for thread in threads:
        thread.start()

    # wait for all threads to finish their work
    running = True
    while running:
        running = False
        for thread in threads:
            if thread.is_alive():
                running = True
                print("waiting for threads to finish...")
                time.sleep(1)
            else:
                if thread.data:
                    dataset.update(thread.data)
                    for d in thread.data:
                        helpers.save_object(thread.data[d], tmp_dir.joinpath(str(d + ".pkl")))

    helpers.save_object(dataset, tmp_dir.joinpath("dataset_full.pkl"))

    # write metadata to csv file
    print("Write dataset to csv")
    f = open(tmp_dir.joinpath("dataset.csv"), "w+")
    f.write("category,name,path,length,length_raw,avgcolumnsize,charratioA,charratioB,year,eyear,lyear,type\r\n")
    for d in dataset:
        for item in dataset[d]:
            f.write("\"" + d + "\",\"" + str(item["metadata"]["name"]) + "\",\"" +
                    str(item["metadata"]["path"]) + "\"," +
                    str(item["metadata"]["length"]) + "," +
                    str(item["metadata"]["length_raw"]) + "," +
                    str(item["metadata"]["avgcolumnsize"]) + "," +
                    str(item["metadata"]["charratioA"]) + "," +
                    str(item["metadata"]["charratioB"]) + ",\"" +
                    str(item["metadata"]["year"]) + "\"," +
                    str(item["metadata"]["eyear"]) + "," +
                    str(item["metadata"]["lyear"]) + ",\"" +
                    str(item["metadata"]["type"]) + "\"\r\n")
    f.close()

    shutil.copyfile(tmp_dir.joinpath("dataset.csv"),
                    models_dir.joinpath("dataset.csv")) # copy dataset.csv to models

    print("data cleaned successfully")
elif "metadata-filtering" not in skip_steps:
    # load dataset_full.pkl because it was not generate during runtime
    dataset = helpers.load_object(tmp_dir.joinpath("dataset_full.pkl"))
    print("data loaded from dataset_full.pkl")

if "metadata-filtering" not in skip_steps:
    for key in data_names_exclude:
        if key in dataset:
            del dataset[key]

    for key in dataset:
        d_tmp = []
        for x in dataset[key]:
            if key == "declaration" and eval(metadata_file_filter_declaration):
                d_tmp.append(x)
            elif eval(metadata_file_filter):
                d_tmp.append(x)
        dataset[key] = d_tmp

    helpers.save_object(dataset, tmp_dir.joinpath("dataset_filtered.pkl"))
    shutil.copyfile(tmp_dir.joinpath("dataset_filtered.pkl"),
                    models_dir.joinpath("dataset_filtered.pkl")) # copy dataset_filtered.pkl to models
elif "modeling" not in skip_steps:
    # load dataset_filtered.pkl because it was not generate during runtime
    dataset = helpers.load_object(tmp_dir.joinpath("dataset_filtered.pkl"))
    print("data loaded from dataset_filtered.pkl")

if "modeling" not in skip_steps:
    t0 = time.time()
    # calculate size of dataset
    dataset_size = 0
    for key in dataset:
        dataset_size = dataset_size + len(dataset[key])

    print("start building corpus")
    BaseManager.register('PoolCorpus', nlp_pool.PoolCorpus)

    if __name__ == '__main__':
        with BaseManager() as manager:
            corp = manager.PoolCorpus()
            corp.set_totalFilesTarget(dataset_size)
            with Pool(processes=number_of_threads) as pool:
                for key in dataset:
                    pool.map(corp.add, ((d["content"], d["metadata"]) for d in dataset[key]))
            corpus = corp.get()
            print("corpus loaded")
            corpus.save(tmp_dir.joinpath("corpus.bin.gz"))
    print("end building corpus")
    print("Time elapsed: ", time.time() - t0, "s")  # CPU seconds elapsed (floating point)

    shutil.copyfile(tmp_dir.joinpath("corpus.bin.gz"),
                    models_dir.joinpath("corpus.bin.gz")) # copy dataset_filtered.pkl to models

# load corpus.bin.gz always because freq cannot handle vanilla corpus
corpus = textacy.Corpus.load("en_core_web_sm", tmp_dir.joinpath("corpus.bin.gz"))
print("data loaded from corpus.bin.gz")

if "analysis_freq" not in skip_steps:
    print("start wordcount")
    # get lowercased and filtered corpus vocabulary (R3.3.1)
    vocab = corpus.word_counts(by='lemma_', filter_stops=True, filter_punct=True, filter_nums=True)
    vocab_doc = corpus.word_doc_counts(by='lemma_', filter_stops=True, filter_punct=True, filter_nums=True)

    # sort vocabulary by descending frequency
    vocab_sorted = sorted(vocab.items(), key=lambda x: x[1], reverse=True)
    vocab_sorted_doc = sorted(vocab_doc.items(), key=lambda x: x[1], reverse=True)

    # write to file, one word and its frequency per line
    with open(tmp_dir.joinpath('vocab_frq.txt'), 'w') as f:
        for word, frq in vocab_sorted:
            line = f"{word}\t{frq}\n"
            f.write(line)
    with open(tmp_dir.joinpath('vocab_frq_doc.txt'), 'w') as f:
        for word, frq in vocab_sorted_doc:
            line = f"{word}\t{frq}\n"
            f.write(line)

    shutil.copyfile(tmp_dir.joinpath("vocab_frq.txt"),
                    analysis_dir.joinpath("vocab_frq.txt")) # copy dataset_filtered.pkl to analysis
    shutil.copyfile(tmp_dir.joinpath("vocab_frq_doc.txt"),
                    analysis_dir.joinpath("vocab_frq_doc.txt"))  # copy dataset_filtered.pkl to analysis

    en = textacy.load_spacy_lang("en_core_web_sm")
    stats = []
    for cat in data_names + ["declaration"]:
        ctmp = corpus.get(lambda doc: doc._.meta["category"] == cat)
        dtmp = list(ctmp)
        ndocs = len(dtmp)
        nvocab = 0
        nlength = 0
        for d in dtmp:
            nvocab = nvocab + d.vocab.length
            nlength = nlength + d._.meta["length"]
        stats.append({"category": cat, "ndocs":  ndocs, "nvocab": nvocab, "nlength": nlength})
        if ndocs > 0:
            tmpcorpus = textacy.corpus.Corpus(en, data=dtmp)
            tmpvocab = tmpcorpus.word_counts(by='lemma_', filter_stops=True, filter_punct=True, filter_nums=True)
            tmpvocab_doc = tmpcorpus.word_doc_counts(by='lemma_', filter_stops=True, filter_punct=True, filter_nums=True)
            tmpvocab_sorted = sorted(tmpvocab.items(), key=lambda x: x[1], reverse=True)
            tmpvocab_sorted_doc = sorted(tmpvocab_doc.items(), key=lambda x: x[1], reverse=True)
            with open(tmp_dir.joinpath('vocab_frq_' + cat + '.txt'), 'w') as f:
                for word, frq in tmpvocab_sorted:
                    line = f"{word}\t{frq}\n"
                    f.write(line)
            with open(tmp_dir.joinpath('vocab_frq_doc_' + cat + '.txt'), 'w') as f:
                for word, frq in tmpvocab_sorted_doc:
                    line = f"{word}\t{frq}\n"
                    f.write(line)
            shutil.copyfile(tmp_dir.joinpath('vocab_frq_' + cat + '.txt'),
                            analysis_dir.joinpath('vocab_frq_' + cat + '.txt'))  # copy dataset_filtered.pkl to analysis
            shutil.copyfile(tmp_dir.joinpath('vocab_frq_doc_' + cat + '.txt'),
                            analysis_dir.joinpath('vocab_frq_doc_' + cat + '.txt'))  # copy dataset_filtered.pkl to analysis
    with open(tmp_dir.joinpath('stats.csv'), 'w') as csv_file:
        writer = csv.DictWriter(csv_file, stats[0].keys())
        writer.writeheader()
        writer.writerows(stats)
    shutil.copyfile(tmp_dir.joinpath('stats.csv'),
                    analysis_dir.joinpath('stats.csv'))  # copy stats.csv to analysis

    print("end wordcount")

if "analysis_advance_preparation" not in skip_steps:
    print("start advance preparation")
    # merge metadata and actual content for each document in the corpus
    # ugly, verbose syntax to merge two dictionaries
    data = [{**doc._.meta, **{'text': doc.text}} for doc in corpus]

    # create panda dataframe
    df = pd.DataFrame(data)

    df_sub = df[(df['text'].str.len() > 10)]

    # make new column containing all relevant metadata (showing in plot later on)
    df_sub['descripton'] = df_sub[['name', 'year', 'charratioB', 'avgcolumnsize']].astype(str).agg(', '.join, axis=1)

    helpers.save_object(df, tmp_dir.joinpath("df.pkl"))
    helpers.save_object(df_sub, tmp_dir.joinpath("df_sub.pkl"))
    print("end advance preparation")

    shutil.copyfile(tmp_dir.joinpath("df.pkl"),
                    models_dir.joinpath("df.pkl")) # copy df.pkl to models
    shutil.copyfile(tmp_dir.joinpath("df_sub.pkl"),
                    models_dir.joinpath("df_sub.pkl")) # copy df_sub.pkl to models
else:
    # load df.pkl and df_sub.pkl because it was not generate during runtime
    df = helpers.load_object(tmp_dir.joinpath("df.pkl"))
    df_sub = helpers.load_object(tmp_dir.joinpath("df_sub.pkl"))
    print("data loaded from df.pkl and df_sub.pkl")

if "analysis_scattertext" not in skip_steps:
    print("start scattertext")
    censor_tags = set(['CARD'])  # tags to ignore in corpus, e.g. numbers

    en = textacy.load_spacy_lang("en_core_web_sm")
    # stop words to ignore in corpus
    en_stopwords = spacy.lang.en.stop_words.STOP_WORDS  # default stop words
    custom_stopwords = set(['[', ']', '%'])
    en_stopwords = en_stopwords.union(custom_stopwords)  # extend with custom stop words

    # create corpus from dataframe
    # lowercased terms, no stopwords, no numbers
    # use lemmas for English only, German quality is too bad
    corpus_speeches = st.CorpusFromPandas(df_sub,  # dataset
                                          category_col='type',  # index differences by ...
                                          text_col='text',
                                          nlp=en,  # EN model
                                          feats_from_spacy_doc=st.FeatsFromSpacyDoc(tag_types_to_censor=censor_tags,
                                                                                    use_lemmas=True),
                                          ).build().get_stoplisted_unigram_corpus(en_stopwords)

    # produce visualization (interactive html)
    html = st.produce_scattertext_explorer(corpus_speeches,
                                           category='declaration',  # set attribute to divide corpus into two parts
                                           category_name='declaration',
                                           not_category_name='textfiles',
                                           metadata=df_sub['descripton'],
                                           width_in_pixels=1000,
                                           minimum_term_frequency=5,  # drop terms occurring less than 5 times
                                           save_svg_button=True,
                                           )

    # write visualization to html file
    fname = tmp_dir.joinpath("viz_declaration_textfiles.html")
    open(fname, 'wb').write(html.encode('utf-8'))
    print("end scattertext")
    shutil.copyfile(tmp_dir.joinpath("viz_declaration_textfiles.html"),
                    analysis_dir.joinpath("viz_declaration_textfiles.html")) # copy viz_declaration_textfiles.html to analysis

if "analysis_year" not in skip_steps:
    print("start year")

    df_sub = df[(df['text'].str.len() > 10)]

    # make new column containing all relevant metadata (showing in plot later on)
    df_sub['descripton'] = df_sub[['name', 'year', 'charratioB', 'avgcolumnsize']].astype(str).agg(', '.join, axis=1)

    dtmp = df_sub.groupby('eyear').agg({'text': "count"}).reset_index().rename(columns={'text': 'count'})
    dtmp = dtmp.rename(columns={"eyear": "year"})
    dtmp.insert(2, "type", "eyear")
    docs_per_year = dtmp

    dtmp = df_sub.groupby('lyear').agg({'text': "count"}).reset_index().rename(columns={'text': 'count'})
    dtmp = dtmp.rename(columns={"lyear": "year"})
    dtmp.insert(2, "type", "lyear")
    docs_per_year = docs_per_year.append(dtmp, ignore_index=True)

    # manual year was only available in top100 analysis
    # dtmp = pd.read_csv('top100_years.txt', delimiter=",").groupby('myear').agg({'text': "count"}).reset_index().rename(
    #    columns={'text': 'count'})
    # dtmp = dtmp.rename(columns={"myear": "year"})
    # dtmp.insert(2, "type", "myear")
    # docs_per_year = docs_per_year.append(dtmp, ignore_index=True)

    docs_per_year = docs_per_year[docs_per_year["year"] != "NA"]
    docs_per_year['year'] = pd.to_numeric(docs_per_year['year'])

    p = (ggplot(docs_per_year, aes('year', 'count', color='type', group='type'))
           + geom_point(alpha=0.5, stroke=0)
           + geom_line()
           + theme_classic()
           + labs(x="Year",
                  y="absolute number",
                  color="Legend")
           + theme(axis_text_x=element_text(angle=90, hjust=1))
           + scale_x_continuous(limits=(1960, 1999))
           )

    ggsave(plot=p, filename="docs_per_year", path=tmp_dir)

    shutil.copyfile(tmp_dir.joinpath("docs_per_year.png"),
                    analysis_dir.joinpath("docs_per_year.png"))  # copy docs_per_year.png to analysis

    try:
        dummy = pd.DataFrame(years.from_1960_to_1999)

        e = dummy.append(docs_per_year[(docs_per_year['type'] == "eyear") &
                                       (docs_per_year['year'] >= 1960)][["year", "count"]])\
            .groupby('year').agg({'count': "sum"})
        l = dummy.append(docs_per_year[(docs_per_year['type'] == "lyear") &
                                       (docs_per_year['year'] >= 1960)][["year", "count"]])\
            .groupby('year').agg({'count': "sum"})

        print("r_{eyear mit lyear} = ", np.corrcoef(e["count"], l["count"])[0, 1])
    except Exception as e:
        print("something went wrong while calculating eyear / lyear variaty.")
        pass

    print("end year")

if "analysis_octis" not in skip_steps:
    print("start octis")

    df_sub.to_csv(tmp_dir.joinpath("corpus.txt"), "\t", columns = ["text"])

    # Initialize preprocessing
    preprocessor = Preprocessing(vocabulary=None, max_features=None,
                                  remove_punctuation=True, punctuation=string.punctuation,
                                  lemmatize=True, stopword_list='english',
                                  min_chars=1, min_words_docs=0)

    # preprocess
    octis_dataset = preprocessor.preprocess_dataset(documents_path=tmp_dir.joinpath("corpus.txt"))

    # save the preprocessed dataset
    octis_dataset.save(str(tmp_dir.joinpath('octis_dataset')))

    if os.path.exists(octis_dataset_dir.joinpath('octis_dataset')):
        shutil.rmtree(octis_dataset_dir.joinpath('octis_dataset'))
    if os.path.exists(models_dir.joinpath('octis_dataset')):
        shutil.rmtree(models_dir.joinpath('octis_dataset'))

    shutil.copytree(tmp_dir.joinpath('octis_dataset'),
                    octis_dataset_dir.joinpath('octis_dataset'))  # copy octis_dataset to datasets
    shutil.copytree(tmp_dir.joinpath('octis_dataset'),
                    models_dir.joinpath('octis_dataset'))  # copy octis_dataset to models

    # octis will be processed using dashboard
    # model = LDA(num_topics=5)  # Create model
    # model_output = model.train_model(octis_dataset)  # Train the model

    # metric = TopicDiversity(topk=10)  # Initialize metric
    # topic_diversity_score = metric.score(model_output)  # Compute score of the metric

    # print("topic diversity score:", topic_diversity_score)

    # wordcloud will be generated outside of python
    # wocl = WordCloud(mode="RGBA", background_color="white").generate(" ".join(model_output["topics"][0]))
    # wocl2 = WordCloud(mode="RGBA", background_color="white", relative_scaling=1, scale=10, max_words=9999,
    #                   min_font_size=1, max_font_size=18, collocations=False).generate(" ".join(model_output["topics"][0]))
    # image = wocl.to_image()
    # image.save(tmp_dir.joinpath("wordcloud.png"))
    # image2 = wocl2.to_image()
    # image2.save(tmp_dir.joinpath("wordcloud2.png"))

    print("end octis")

if "analysis_entities" not in skip_steps:
    entities = []

    for doc in corpus.docs:
        for ent in textacy.extract.entities(doc):
            try:
                entities += [{"text": ent.text, "label": ent.label_, "explain": spacy.explain(ent.label_)}]
            except:
                print("Problem with:", doc._.meta["name"])

    # export corpus as csv
    f_csv = tmp_dir.joinpath('entities.csv')
    textacy.io.csv.write_csv(entities, f_csv, fieldnames=entities[0].keys())

    shutil.copyfile(tmp_dir.joinpath('entities.csv'),
                    analysis_dir.joinpath('entities.csv'))  # copy entities.csv to analysis

    df_entities = pd.DataFrame(entities, columns=['text', 'label', 'explain'])
    df_entities_count = df_entities.groupby('text').agg({'label': "count"}).rename(
        columns={'label': 'count'}).sort_values(by=['count'], ascending=False).reset_index()

    # write to file, one word and its frequency per line
    fname = tmp_dir.joinpath('entities_frq.csv')
    with open(fname, 'w') as f:
        for i, d in df_entities_count.iterrows():
            line = d["text"] + "," + str(d["count"]) + "\n"
            f.write(line)

    shutil.copyfile(tmp_dir.joinpath('entities_frq.csv'),
                    analysis_dir.joinpath('entities_frq.csv'))  # copy entities_frq.csv to analysis

    print("entities: \n", df_entities_count[:25])

print("everything done.")
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

############################################################
# title:  datawrangler
# author: Josias Bruderer
# date:   27.08.2021
# desc:   this module takes care of all tasks that are
#         related to juggling files and datasets.
############################################################

import os
import sys
from threading import Thread
from pathlib import Path
import requests
import zipfile
import codecs
import re


def averageLen(lst, excludeEmpty=True):
    if excludeEmpty:
        lengths = [len(i) for i in lst if i != "" and i != " "]
    else:
        lengths = [len(i) for i in lst]
    return 0 if len(lengths) == 0 else round((float(sum(lengths)) / len(lengths)), 2)


def daterange(lst, t="r"):
    ltmp = []
    ltmp2 = []
    if len(lst) > 0:
        for l in lst:
            ltmp += list(filter(None, l))
        for l in ltmp:
            if len(l) == 2:
                ltmp2 += ["19" + l]
            else:
                ltmp2 += [l]
        if len(ltmp2) > 2:
            if t == "e":
                return str(min(ltmp2))
            elif t == "l":
                return str(max(ltmp2))
            else:
                return str(min(ltmp2) + "-" + max(ltmp2))
        else:
            return str(ltmp2[0])
    else:
        return "NA"

class cleaner(Thread):

    def __init__(self, data_dir, data_names, file_filter):
        Thread.__init__(self)
        self.data_dir = data_dir
        self.data_names = data_names
        self.file_filter = file_filter
        self.data = {}

    def run(self):
        for data_name in self.data_names:
            self.data[data_name] = self.get_texts(Path(self.data_dir, data_name), data_name)

    def get_texts(self, dir_texts, data_name):
        """
        Sequentially stream all documents from a given folder,
        including metadata.
        """
        data = []

        # iterate over all documents
        for fname in dir_texts.glob('**/*'):  # ** = all subdirectories
            if Path(fname).is_file():
                print("processing in " + str(dir_texts.stem) + " file: " + str(fname))

                if re.match(self.file_filter, fname.name):
                    print("skip in " + str(dir_texts.stem) + " because of file_filter: " + str(fname))
                    continue
                # Read file content and replace encoding erros
                content_raw = codecs.open(fname, 'r', encoding='utf-8', errors='replace').read()

                # join lines as there are hard line-breaks
                content = content_raw.replace('\r\n', ' ')
                content = content.replace('\r', ' ')
                content = content.replace('\n', ' ')
                content = content.replace('\t', ' ')
                content = content.replace('\x1a', ' ')
                content = re.sub('[^A-z0-9\ \.\'\,\!]', ' ', content)
                content = re.sub('[\\\\\^\[\]]', ' ', content)

                # add more metadata here if needed
                # charratio: 0 = no character is "text", 1 = every character is "text"
                if len(content_raw) == 0:
                    charratioA = 0
                    charratioB = 0
                else:
                    charratioA = round(1 / len(content_raw) * len(re.findall("[A-z]", content_raw)), 2)
                    charratioB = round(1 / len(content_raw) * len(re.findall("[A-z\ \.\"\,\!]", content_raw)), 2)

                typ = "textfile"
                if fname.name == "declarationbarlow1996.txt" or data_name == "declaration":
                    typ = "declaration"

                rxdate = re.compile(
                    'copyright.{0,3}(19[6-9][0-9])|updated.{0,3}[0-1]?[0-9]?-[0-3]?[0-9]?-([6-9][0-9])|'
                    'Date\:.*([6-9][0-9]).*,|(?:jan(?:uary)?|feb(?:ruary)?|mar(?:ch)?|apr(?:il)?|may|'
                    'june|july|aug(?:ust)?|sept(?:ember)?|oct(?:ober)?|nov(?:ember)?|dec(?:ember)?)'
                    '.{0,8}(1?9?[6-9][0-9])|[0-1]?[0-9]?\/[0-3]?[0-9]?\/([6-9][0-9])|[0-1]?[0-9]?-[0-3]?[0-9]?'
                    '-([6-9][0-9])|[^-](19[6-9][0-9])')

                matches = rxdate.findall(content, re.IGNORECASE)

                metadata = {'name': fname.name,
                            'path': str(fname),
                            'length_raw': len(content_raw),
                            'length': len(content),
                            'avgcolumnsize': averageLen(content_raw.splitlines()),
                            'charratioA': charratioA,
                            'charratioB': charratioB,
                            'year': daterange(matches),
                            'eyear': daterange(matches, "e"),
                            'lyear': daterange(matches, "l"),
                            'type': typ,
                            'category': data_name,
                            }

                # return documents one after another (sequentially)
                data.append({"content": content, "metadata": metadata})
        return data

class loader(Thread):

    def __init__(self, tmp_dir, data_dir, data_url, data_names):
        Thread.__init__(self)
        self.tmp_dir = tmp_dir
        self.data_dir = data_dir
        self.data_url = data_url
        self.data_names = data_names
        self.init()

    def init(self):
        try:
            # create tmp directory if not existing yet
            if not os.path.exists(self.tmp_dir.is_dir()):
                os.mkdir(self.tmp_dir)

            # create data directory if not existing yet
            if not os.path.exists(self.data_dir):
                os.mkdir(self.data_dir)
        except:
            print("Unexpected error: ", sys.exc_info()[0])
            raise

    def run(self):
        for data_name in self.data_names:
            url = self.data_url.replace("[name]", data_name)
            zipdir = Path(self.tmp_dir, str(data_name + ".zip"))
            self.download_zip(url, zipdir)
            self.extract_zip(zipdir)

    def download_zip(self, url, zipdir):
        try:
            if not zipdir.is_file():
                print("Downloading: " + url)
                r = requests.get(url, allow_redirects=True)
                open(zipdir, 'wb').write(r.content)
            else:
                print("Skip downloading, file already downloaded: " + url)
        except:
            print("Unexpected error: ", sys.exc_info()[0])
            raise

    def extract_zip(self, zipdir):
        try:
            if not Path(self.data_dir / zipdir.stem).is_dir():
                print("Extracting: " + str(zipdir))
                with zipfile.ZipFile(zipdir, 'r') as zip_ref:
                    zip_ref.extractall(self.data_dir)
            else:
                print("Skip extracting, file already extracted: " + str(zipdir))
        except:
            print("Unexpected error: ", sys.exc_info()[0])
            raise

```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

##########################################################
# title:  helpers
# author: Josias Bruderer
# date:   25.08.2021
# desc:   this module provides some useful functions
##########################################################

import pickle


def save_object(obj, filename):
    with open(filename, 'wb') as outp:  # Overwrites any existing file.
        pickle.dump(obj, outp, pickle.HIGHEST_PROTOCOL)

def load_object(filename):
    with open(filename, 'rb') as inp:
        return pickle.load(inp)

def chunker_list(seq, size):
    return (seq[i::size] for i in range(size))

```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

##########################################################
# title:  helpers
# author: Josias Bruderer
# date:   28.08.2021
# desc:   this module provides nlp functions
##########################################################

import textacy
import spacy
# run: ./.envs/bin/python -m spacy download en_core_web_sm


class PoolCorpus(object):

    def __init__(self):
        model = spacy.load('en_core_web_sm', disable=["parser"])
        model.max_length = 10000000  # enable utilization of ~ 100GB RAM
        self.corpus = textacy.corpus.Corpus(lang=model)
        self.totalFilesTarget = 1
        self.processedFiles = 0

    def add(self, data):
        self.corpus.add(data)
        self.processedFiles = self.processedFiles + 1
        print("Processed ", self.processedFiles, " of ", self.totalFilesTarget, "files: ",
              round(100/self.totalFilesTarget*self.processedFiles, 4), "%")

    def get(self):
        return self.corpus

    def save(self, path):
        self.corpus.save(path)

    def set_totalFilesTarget(self, n):
        self.totalFilesTarget = n

"""
texts = {
        'key1': 'First text 1.',
        'key2': 'Second text 2.',
        'key3': 'Third text 3.',
        'key4': 'Fourth text 4.',
    }

BaseManager.register('PoolCorpus', PoolCorpus)

if __name__ == '__main__':
    with BaseManager() as manager:
        corpus = manager.PoolCorpus()

        with Pool(processes=2) as pool:
            pool.map(corpus.add, ((v, {'key': k}) for k, v in texts.items()))

        print(corpus.get())
"""
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

##########################################################
# title:  helpers
# author: Josias Bruderer
# date:   29.08.2021
# desc:   this module provides years variable
##########################################################

from_1960_to_1999 = [{"year": 1960, "count": 0},
                      {"year": 1961, "count": 0},
                      {"year": 1962, "count": 0},
                      {"year": 1963, "count": 0},
                      {"year": 1964, "count": 0},
                      {"year": 1965, "count": 0},
                      {"year": 1966, "count": 0},
                      {"year": 1967, "count": 0},
                      {"year": 1968, "count": 0},
                      {"year": 1969, "count": 0},
                      {"year": 1970, "count": 0},
                      {"year": 1971, "count": 0},
                      {"year": 1972, "count": 0},
                      {"year": 1973, "count": 0},
                      {"year": 1974, "count": 0},
                      {"year": 1975, "count": 0},
                      {"year": 1976, "count": 0},
                      {"year": 1977, "count": 0},
                      {"year": 1978, "count": 0},
                      {"year": 1979, "count": 0},
                      {"year": 1980, "count": 0},
                      {"year": 1981, "count": 0},
                      {"year": 1982, "count": 0},
                      {"year": 1983, "count": 0},
                      {"year": 1984, "count": 0},
                      {"year": 1985, "count": 0},
                      {"year": 1986, "count": 0},
                      {"year": 1987, "count": 0},
                      {"year": 1988, "count": 0},
                      {"year": 1989, "count": 0},
                      {"year": 1990, "count": 0},
                      {"year": 1991, "count": 0},
                      {"year": 1992, "count": 0},
                      {"year": 1993, "count": 0},
                      {"year": 1994, "count": 0},
                      {"year": 1995, "count": 0},
                      {"year": 1996, "count": 0},
                      {"year": 1997, "count": 0},
                      {"year": 1998, "count": 0},
                      {"year": 1999, "count": 0}]
```