Check fo duplicate articles and data objects

A list of publications is obtainded from the app database. This list will contain a titles, IDs and DOIs which need to be explored to look for duplicates. The steps of the process are:

1. get a Title, DOI, and URL for each publication
2. revise each element of the list for duplicates

In [ ]:
```
# Libraries
# library containign functions that read and write to csv files
import lib.handle_csv as csvh
# library for connecting to the db
import lib.handle_db as dbh
# library for handling text matchings
import lib.text_comp as txtc
# library for getting data from crossref
import lib.crossref_api as cr_api
# library for handling url searchs
import lib.handle_urls as urlh
# managing files and file paths
from pathlib import Path
# add aprogress bar
from tqdm import notebook
#library for handling json files
import json
# library for using regular expressions
import re
# library for handling http requests
import requests
# import custom functions (common to various notebooks)
import processing_functions as pr_fns

# datetime parsing
from datetime import datetime

current_step = 1
```

Verify if there are duplicates in articles

1. Open the current publication list from the appdb
2. read each entry and check if there are duplicates in doi, url or title

In [ ]:
```
# deal with no pdf_file column
def get_pubs_list(db_path):
    pubs_list = None
    try:
        pubs_list = pr_fns.get_pub_data(db_path)
    except Exception as inst:
        if 'pdf_file' in inst.args[0]:
            print('problem articles table does not have pdf_file column')
            pass
    try:
        if pubs_list == None:
            pubs_list = pr_fns.get_pub_app_data(db_path)
    except Exception as inst:
        print(type(inst))
        print(inst.args)
        print(inst)
        print('another problem')
    return pubs_list

# 1 current app DB
db_name = 'production'
#ukchapp_db = "db_files/"+db_name+".sqlite3"
ukchapp_db = "../mcc_data/"+db_name+".sqlite3"
ukchapp_db = "./db_files/"+db_name+".sqlite3"
while not Path(ukchapp_db).is_file():
    print('Please enter the name of app db file:')
    ukchapp_db = input()

#  get publication data from the ukch app
app_pubs = get_pubs_list(ukchapp_db)
```

In [ ]:

```
# 2 read each entry and check if there are duplicates in doi, url or title
dup_list={}
dup_count = 0
if current_step == 1 and app_pubs != None:
    dups = []
    for idx, a_pub in enumerate(notebook.tqdm(app_pubs)):
        pub_id = a_pub[0]
        pub_title = a_pub[1]
        pub_doi = a_pub[2]
        pub_link = a_pub[3]
        # verfy if dois are duplicated
        if pub_doi != None:
            for i_indx in range(idx+1, len(app_pubs)):
                #print(pub_doi, app_pubs[i_indx][2])
                if app_pubs[i_indx][2]!=None and pub_doi.strip().lower() ==  app_pubs[i_ind
                    print("\nDOI", pub_doi, "duplicated at:", i_indx, app_pubs[i_indx], app
                    a_dup = {'pub_comp': app_pubs[idx], 'pub_dup': app_pubs[i_indx],"dup_at
                    dup_count+=1
                    dup_list[dup_count] = a_dup
        # verify if urls are all unique
        if pub_link != None:
            for i_indx in range(idx+1, len(app_pubs)):
                if app_pubs[i_indx][3]!=None and pub_link.strip().lower() ==  app_pubs[i_in
                    print("\nLink", pub_url, "duplicated at:", i_indx, app_pubs[i_indx], ap
                    a_dup = {'pub_comp': app_pubs[idx], 'pub_dup': app_pubs[i_indx],"dup_at
                    dup_count+=1
                    dup_list[dup_count] = a_dup
        # verify if titles are all unique
        if pub_title != None:
            for i_indx in range(idx+1, len(app_pubs)):
                similarity = txtc.similar(pub_title.strip().lower(), app_pubs[i_indx][1].st
                #print(similarity)
                if app_pubs[i_indx][1]!=None and pub_title.strip().lower() ==  app_pubs[i_i
                    print("\nTitle", pub_title, "duplicated at:", i_indx,app_pubs[i_indx][1
                    print("Similarity:", similarity)
                    a_dup = {'pub_comp': app_pubs[idx], 'pub_dup': app_pubs[i_indx],"dup_at
                    dup_count+=1
                    dup_list[dup_count] = a_dup
                elif similarity > 0.8:
                    print("Title", similarity, ":\n\t", pub_id, pub_title, "\nSimilar to:\n
if len(dup_list) > 0:
        csvh.write_csv_data(dup_list, 'dup_'+db_name+'.csv')
else:
    print ("No duplicate articles in DB")
```

Verify if there are duplicates in data objects

1. get the current data objects list from the appdb
2. read each entry and check if there are duplicates in doi, url or title

In [ ]:

```
# 1 current app DB
dup_list={}
#db_name = 'development'
#ukchapp_db = "db_files/"+db_name+".sqlite3"
#ukchapp_db = "../mcc_data/"+db_name+".sqlite3"
while not Path(ukchapp_db).is_file():
    print('Please enter the name of app db file:')
    ukchapp_db = input()

#  get datasets list from the ukch app
app_datasetes = pr_fns.get_dataset_data(ukchapp_db)

# 2 read each entry and check if there are duplicates in doi, url or title
if current_step == 1:
    dup_list={}
    dup_count = 0
    for idx, a_ds in enumerate(notebook.tqdm(app_datasetes)):
        ds_id = a_ds[0]
        ds_doi = a_ds[1]
        ds_url = a_ds[2]
        ds_name = a_ds[3]
        #print (ds_id, ds_doi, ds_url, ds_name)
        # verfy if dois are duplicated
        if ds_doi != None and ds_doi != '':
            for i_indx in range(idx+1, len(app_datasetes)):
                #print(pub_doi, app_pubs[i_indx][1])
                if app_datasetes[i_indx][1]!=None and ds_doi.strip().lower() ==  app_datase
                    print("\nDuplicate DOI found",
                          "\nDO ID:", ds_id, "Title:", ds_name, "\nDOI:", ds_doi,
                          "\nDO ID:", app_datasetes[i_indx][0], "Title:",
                          app_datasetes[i_indx][3], "\nDOI:", app_datasetes[i_indx][1])
                    a_dup = {'pub_comp': app_datasetes[idx], 'pub_dup': app_datasetes[i_ind
                    dup_count+=1
                    dup_list[dup_count] = a_dup
        # verify if urls are all unique, if doi not equal it is OK, in some cases
        if ds_url != None:
            for i_indx in range(idx+1, len(app_datasetes)):
                #print(app_datasetes[i_indx])
                if app_datasetes[i_indx][2]!=None and ds_url.strip().lower() ==  app_datase
                    print("\nDuplicate found URL:",
                          "\nDO ID:", ds_id, "Title:", ds_name, "\nURL:", ds_url,
                          "\nDO ID:", app_datasetes[i_indx][0], "Title:",
                          app_datasetes[i_indx][3], "\nURL:", app_datasetes[i_indx][2])
                    a_dup = {'pub_comp': app_datasetes[idx], 'pub_dup': app_datasetes[i_ind
                    dup_count+=1
                    dup_list[dup_count] = a_dup
        # verify if titles are all unique, if doi not equal it is OK in some cases
        if ds_name != None:
            for i_indx in range(idx+1, len(app_datasetes)):
                similarity = txtc.similar(ds_name.strip().lower(), app_datasetes[i_indx][3]
                #print(similarity)
                if app_datasetes[i_indx][1]!=None and ds_name.strip().lower() ==  app_datas
                    if app_datasetes[i_indx][1] != None and app_datasetes[idx][1] != None a
                    app_datasetes[i_indx][1] == app_datasetes[idx][1]:
                        print("\nDuplicate found Similarity:", similarity,
                              "\nDO ID:", ds_id, "Title:", ds_name,"\nDO ID:",
                              app_datasetes[i_indx][0], "Title:", app_datasetes[i_indx][3])
                        a_dup = {'pub_comp': app_datasetes[idx], 'pub_dup': app_datasetes[i
                        dup_count+=1
                        dup_list[dup_count] = a_dup
                #elif similarity > 0.8:
                #    print(similarity, "Title:\n\t", ds_name, "\n\t- similar at:\n\t", i_in
                #    break
if len(dup_list) > 0:
```

```
    csvh.write_csv_data(dup_list, 'dup_do_'+db_name+'.csv')
    print("duplicates saved to",'dup_do_'+db_name+'.csv')
else:
    print ("No duplicate DOs in DB")
```

Verify authors

1. verify that there are no authors with no articles in the DB
2. verify that authors are unique remove close matches (need to check spellings)

In [ ]:
```
def make_like_str(a_string):
    like_str = "%" + re.sub(r'[^a-zA-Z\s:]', '%', a_string) + "%"
    return like_str

def get_all_authors():
    db_conn = dbh.DataBaseAdapter(ukchapp_db)
    s_table = 'authors'
    s_fields = 'id, given_name, last_name, orcid'
    s_where = 'isap IS NULL' # 1 displayed authors - 0/NULL the rest
    authors_list = db_conn.get_values(s_table, s_fields, s_where)
    return authors_list

def get_null_authors():
    db_conn = dbh.DataBaseAdapter(ukchapp_db)
    s_table = 'authors'
    s_fields = 'id, given_name, last_name, orcid'
    s_where = 'id NOT IN (SELECT article_authors.author_id FROM article_authors)'
    authors_list = db_conn.get_values(s_table, s_fields, s_where)
    return authors_list

def get_similar_authors(name,surname,orcid):
    db_conn = dbh.DataBaseAdapter(ukchapp_db)
    s_table = 'authors'
    s_fields = 'id, given_name, last_name, orcid'
    like_surname = make_like_str(surname)
    surname.replace("'","''")
    s_where = "(orcid = '%s' AND last_name = '%s')"%(orcid, surname)
    s_where += "OR(given_name = '%s' AND last_name = '%s')"%(name, surname)
    s_where += "OR(last_name = '%s')"%(surname)
    s_where += "OR(last_name LIKE '%s')"%(like_surname)
    authors_list = db_conn.get_values(s_table, s_fields, s_where)
    return authors_list

def count_linked(author_id):
    db_conn = dbh.DataBaseAdapter(ukchapp_db)
    s_table = 'article_authors'
    s_fields = 'id, author_id'
    s_where = "(author_id = %s)"%(author_id)
    aa_list = db_conn.get_values(s_table, s_fields, s_where)
    return len(aa_list)
```

In [ ]:
```
from IPython.display import clear_output

def save_ok_list(values_list, file_name):
    with open(file_name, 'w') as f:
        for an_id in values_list:
            f.write(str(an_id)+'\n')

def open_ok_list(file_name):
    with open(file_name) as f:
        lines = f.readlines()
    from_file = []
    for a_line in lines:
        from_file.append(int(a_line.replace('\n','')))
    return from_file

def add_to_ok_list(a_value, file_name):
    with open(file_name, 'a') as f:
        f.write(str(a_value)+'\n')
```

In [ ]:
```
def is_single_word(a_word, another_word):
    single_word = False
    in_word = another_word.lower().find(a_word.lower())
    if in_word >= 0:
        single_word = True
        if in_word > 0 and  another_word[in_word-1].isalpha():
            sinlge_word = False
        if in_word + len(a_word) < len(another_word)-1 and another_word[in_word + len(a_wor
            single_word = False
    return single_word

def prune_similar_surnames(the_similars, a_surname):
    pruned_list = []
    for a_simi in the_similars:
        if a_simi[2] == a_surname or is_single_word(a_surname,a_simi[2]) :
            pruned_list.append(a_simi)
    return pruned_list

def get_initials(given_names):
    initials_1 =  [a_letter for a_letter in given_names if a_letter.isupper() ]
    names = given_names.split()
    initials_2 = [a_name[0] for a_name in given_names]
    ri_1 = ". ".join(initials_1)+"."
    ri_2 = " ".join(initials_1)
    return ri_1, ri_2

def prune_similar_names(the_similars, a_name):
    pruned_list = []
    dot_initials, initials = get_initials(a_name)
    for a_simi in the_similars:
        if a_simi[1] == a_name or is_single_word(a_name,a_simi[1]) :
            pruned_list.append(a_simi)
        elif initials == a_simi[1] or dot_initials == a_simi[1]:
            pruned_list.append(a_simi)
    return pruned_list
```

In [ ]:
```
def set_author_value(a_id, a_column, a_value):
    db_conn = dbh.DataBaseAdapter(ukchapp_db)
    db_conn.set_value_table('authors', a_id, a_column, a_value)


def update_author(old_author, a_id, a_name, a_surname, a_orcid):
    if a_id != old_author[0]:
        return # do not update different authors
    if a_name != old_author[1]:
        set_author_value(a_id, 'given_name', a_name)
    if a_surname != old_author[2]:
        set_author_value(a_id, 'last_name', a_surname)
    if a_orcid != old_author[3]:
        set_author_value(a_id, 'given_name', a_orcid)

def update_article_authors(new_id, old_id):
    db_conn = dbh.DataBaseAdapter(ukchapp_db)
    s_where = "author_id = %s" % (old_id)
    aa_ids = db_conn.get_values('article_authors', 'id', s_where)
    print(aa_ids)
    for an_id in aa_ids:
        db_conn.set_value_table('article_authors', an_id[0], 'author_id', new_id)

def delete_author(a_id):
    db_conn = dbh.DataBaseAdapter(ukchapp_db)
    db_conn.connection.execute("DELETE FROM authors WHERE id = %s" % (a_id ))
    db_conn.connection.commit()

def merge_authors(an_author, auth_similars):
    auth_id = an_author[0]
    auth_name = an_author[1]
    auth_surname = an_author[2]
    auth_orcid = an_author[3]

    for a_result in auth_similars:
        if auth_id > a_result[0]:
            auth_id = a_result[0]
        if auth_name != a_result[1] and len(auth_name) < len(a_result[1]):
            print("Which name \n\t 1)", auth_name, "\n\t 2)", a_result[1])
            opt_name = input()
            if opt_name == '2': auth_name = a_result[1]
        if auth_surname != a_result[2] and len(auth_surname) < len(a_result[2]):
            print("Which surname \n\t 1)", auth_surname, "\n\t 2)", a_result[2])
            opt_name = input()
            if opt_name == '2': auth_surname = a_result[2]
        if auth_orcid != a_result[3]:
            print("Which ORCID \n\t 1)", auth_orcid, "\n\t 2)", a_result[3])
            opt_name = input()
            if opt_name == '2': auth_orcid = a_result[3]

    if auth_id != an_author[0] or auth_name != an_author[1] or auth_surname != an_author[2]
        print ("Will update", an_author, "to", auth_id , auth_name , auth_surname,auth_orci

    for a_result in auth_similars:
        if auth_id != a_result[0]:
            print("will update all author articles from:", a_result[0], "to:", auth_id)
            print("will delete author:", a_result[0])
    print("Continue?\n\t 1) proceed \n\t 2) cancel")
    opt_go = input()
    if opt_go == '1':
        update_author(an_author, auth_id, auth_name , auth_surname,auth_orcid)
        for a_result in auth_similars:
            if auth_id != a_result[0]:
                update_article_authors(auth_id,a_result[0])
                delete_author(a_result[0])
```

Check authors with no articles

In [ ]:
```
# verify that there are no authors with no articles in the DB

no_artaut_authors = get_null_authors()

delete_these=[]
for an_author in notebook.tqdm(no_artaut_authors):
    dup_id = an_author[0]
    dup_name = an_author[1]
    dup_surname = an_author[2]
    dup_orcid = an_author[3] if an_author[3]!=None else "NULL"
    print(dup_id, dup_name, dup_surname, dup_orcid)
    similars = get_similar_authors(dup_name, dup_surname, dup_orcid)
    print('There are %s similar authors in DB'%(len(similars)))
    for idx, a_simil in enumerate(similars):
        art_count = count_linked(a_simil[0])
        if art_count == 0:
            print(idx, a_simil,"Links:", art_count, "DELETE")
            delete_these.append(a_simil[0])
        else:
            print(idx, a_simil,"Links:", art_count, "CHECK")

print(delete_these)
```

Review duplicate authors (by name and last name)

In [ ]:
```
# Manually review probable duplicate authors
all_authors = get_all_authors()
revise_these=[]

safe_list = open_ok_list('safe_list.txt')
pacer_idx = 0
for an_author in notebook.tqdm(all_authors):
    dup_id = an_author[0]
    dup_name = an_author[1]
    dup_surname = an_author[2]
    dup_orcid = an_author[3] if an_author[3]!=None else "NULL"
    if not(int(dup_id) in safe_list):
        if "'" in dup_surname: dup_surname = dup_surname.replace("'","''")
        if ("’") in dup_surname: dup_surname = dup_surname.replace("’","''")
        all_similars = get_similar_authors(dup_name, dup_surname, dup_orcid)
        similars = prune_similar_surnames(all_similars, dup_surname)
        similars = prune_similar_names(similars, dup_name)
        if len(similars) > 1 and  len(dup_surname)>3 :
            print("************************************************")
            print("Author:", dup_id, dup_name, dup_surname, dup_orcid)
            print('There are %s similar authors in DB'%(len(similars)))

            for idx, a_simil in enumerate(similars):
                art_count = count_linked(a_simil[0])
                print(idx, a_simil,"Links:", art_count)
            print ("Options:\n\t (1) Ignore \n\t (2) Merge\n\t (3) next")
            sel_action = input()
            if sel_action == '1':
                safe_list = sorted(list(set(safe_list + [sublist[0] for sublist in similars
            if sel_action == '2':
                merge_authors(an_author, similars)
            pacer_idx+=1
            clear_output()
            if pacer_idx == 10:
                break
print("OK:", len(safe_list))
print(safe_list)
save_ok_list(safe_list, 'safe_list.txt')
```

Verify affiliations and author affiliations agains crossref affiliations

1 get group of crossref affiliations 2 get assigned affiliation 3 verify if OK if not show and ask for action

In [ ]:
```
import craffiparser

def get_parser(db_):
    cr_parse = craffiparser.crp(db_)
    cr_parse.start_lists()
    return cr_parse


def get_cr_affis_article_author_ids(db_name):
    db_conn = dbh.DataBaseAdapter(db_name)
    a_table = 'cr_affiliations'
    a_column = 'article_author_id'
    cr_affis_article_author_ids = db_conn.get_value_list(a_table, a_column)
    return cr_affis_article_author_ids

def get_cr_lines_for_article_author_ids(db_name, art_author_id):
    db_conn = dbh.DataBaseAdapter(db_name)
    s_table = 'cr_affiliations'
    s_fields = '*'
    s_where = "article_author_id = %s"%(art_author_id)
    authors_list = db_conn.get_values(s_table, s_fields, s_where)
    return authors_list

def get_affiliation_id(db_name, parsed_affi):
    db_conn = dbh.DataBaseAdapter(db_name)
    s_table = 'affiliations'
    s_field = 'id'
    for k,v in parsed_affi.items():
        if "'" in v :parsed_affi[k]=v.replace("'","''")
    list_where = [ k +" = '"+ v +"'" for k,v in parsed_affi.items() if k != 'address']
    s_where = " AND ".join(list_where)
    s_where = s_where.replace("= ''", "IS NULL")
    print (s_where)
    affi_list = db_conn.get_values(s_table, s_field, s_where)
    affi_id = None
    if affi_list !=[]:
        affi_id = affi_list[0][0]
    return affi_id

# could correct the close affiliation to get all the ones with
# same institution and compare closest match
def get_close_affiliation_id(db_name, parsed_affi):
    db_conn = dbh.DataBaseAdapter(db_name)
    s_table = 'affiliations'
    s_field = 'id'
    for k,v in parsed_affi.items():
        if "'" in v :parsed_affi[k]=v.replace("'","''")
    list_where = [ k +" = '"+ v +"'" for k,v in parsed_affi.items() if k != 'address']
    s_where = " AND ".join(list_where)
    s_where = s_where.replace("= ''", "IS NULL")
    #print (s_where)
    affi_list = db_conn.get_values(s_table, s_field, s_where)
    affi_id = None
    if affi_list !=[]:
        affi_id = affi_list[0][0]
    return affi_id

#get the id of affiliation assigned to an author affiliation record
def get_auth_affi_affiliation_id(db_name, aut_affi_id):
    db_conn = dbh.DataBaseAdapter(db_name)
    s_table = 'author_affiliations'
    s_field = 'affiliation_id'
    s_where = " id = %i" %(aut_affi_id)
    #print (s_where)
```

```
    affi_list = db_conn.get_values(s_table, s_field, s_where)
    if affi_list !=[]:
        affi_list = list(set([an_id[0] for an_id in affi_list]))
    return affi_list

#get the ids the author affiliation records for a given author
def get_auth_affi_id_for_author(db_name, art_aut_id):
    db_conn = dbh.DataBaseAdapter(db_name)
    s_table = 'author_affiliations'
    s_field = 'id'
    s_where = " article_author_id = %i" %(art_aut_id)
    #print (s_where)
    affi_list = db_conn.get_values(s_table, s_field, s_where)
    if affi_list !=[]:
        affi_list = list(set([an_id[0] for an_id in affi_list]))
    return affi_list

def is_one_line_affi(cr_parser, str_affi):
    is_one_liner = False
    parsed_affi = cr_parser.split_single(str_affi)
    parsed_no_blanks = {k:v for k,v in parsed_affi.items() if v != ''}
    if len(parsed_no_blanks) > 1:
        is_one_liner = True
    return is_one_liner

def check_assigned_affi_ol(db_name, cr_parser, cr_affi):
    assigned_ok = False
    if cr_affi[3] == -1:
        assigned_ok = True
        print('non assigned line for', str(cr_affi))
    elif cr_affi[3] != None:
        parsed_affi = cr_parser.split_single(cr_affi[1])
        parsed_no_blanks = {k:v for k,v in parsed_affi.items() if v != ''}
        affi_id = get_affiliation_id(db_name, parsed_affi)
        if affi_id == None:
            affi_id = get_close_affiliation_id(db_name, parsed_no_blanks)
        ###############################################################################
        # if there is no close affiliation should ask if add, assign or ignore
        # in the case of orphan lines it is ignore
        print(cr_affi)
        assigned_affi_id = get_auth_affi_affiliation_id(db_name, cr_affi[3])[0]

        print('Assigned ID:', assigned_affi_id, "Recoverd ID:", affi_id)

        if assigned_affi_id == affi_id:
            assigned_ok = True
    return assigned_ok

def check_assigned_affi_ml(db_name, cr_parser, cr_affi_lines, art_aut_id):
    assigned_ok = True
    just_affi_lines = [x[1] for x in cr_affi_lines]
    parsed_affis = cr_parser.parse_multiline(just_affi_lines)
    # all affiliations belong to same article author
    aut_affis = get_auth_affi_id_for_author(db_name, art_aut_id)
    assigned_affis = []
    for an_aut_affi_id in aut_affis:
        assigned_affis.append(get_auth_affi_affiliation_id(db_name, an_aut_affi_id)[0])

    for one_parsed in parsed_affis:
        affi_id = get_affiliation_id(db_name, one_parsed)
        if affi_id == None:
            parsed_no_blanks = {k:v for k,v in one_parsed.items() if v != ''}
            affi_id = get_close_affiliation_id(db_name, parsed_no_blanks)
        # if there is no close affiliation should ask if add, assign or ignore
        # in the case of orphan lines it is ignore
        if not affi_id in assigned_affis:
            print('Assigned ID:', affi_id, "not in recoverd IDs list:", assigned_affis)
            assigned_ok = False
        else:
            print('Assigned ID:', affi_id, "in recoverd IDs list:", assigned_affis)
    return assigned_ok

###############################################################################
# FIX AFFILIATION ISSUES
# Likely problems:
#   a) only one assigned to two affiliations
#      Fixes:
#        - add missing author affiliation
#        - correct exiting author affiliation
#   b) Mismatch in assigned affiliation
#      Fixes:
#        - correct exiting author affiliation
#   c) Affiliation not assigned
#      Fixes:
#        - try to assign from existing
#        - if no existing one, ask if new should be added

def correct_oneline(db_name, cr_parser, cr_affis):
    # get a list of parsed affis with the ids of the corresponding cr_records
    parsed_affis  =[]
    for a_cr_affi in cr_affis:
        parsed_affis += cr_parser.parse_and_map_single(a_cr_affi)
    print(parsed_affis)
    # all belong to same article author
    art_author_id = cr_affis[0][2]

    print ("verifying affiliations for article author", art_author_id)

    art_auth_affis = get_auth_affi_id_for_author(db_name, art_author_id)

    print ("Article author affiliations:", len(art_auth_affis), art_auth_affis )

    print ("Parsed article author affiliations:", len(parsed_affis) )

    for affi_idx, parsed_affi in enumerate(parsed_affis):
        print('processing', parsed_affi)
        affi_vals = parsed_affi[0]
        cr_affi_ids = parsed_affi[1]
        correct_this = 0
        if affi_idx < len(art_auth_affis):
            correct_this = art_auth_affis[affi_idx]#
        affi_id = get_affiliation_id(db_name, affi_vals)
        if affi_id == None:
            parsed_no_blanks = {k:v for k,v in affi_vals.items() if v != ''}
            affi_id = get_close_affiliation_id(db_name, parsed_no_blanks)
        if correct_this != 0:
            # the affiliation does not exist but something was assigned to author affi
            if affi_id == None:
                print('{0:*^80}'.format('Affi does not exist'))
                print(affi_vals)
                affi_id = add_new_affiliation(db_name, affi_vals)
            # if the affiliation exists
            if affi_id != None:
                print('{0:*^80}'.format('Update Author Affiliatio'))
                print('Update ID:', correct_this, 'with values:', affi_vals )
                update_author_affiliation(db_name, correct_this, affi_id, affi_vals)
                update_cr_aai(db_name, cr_affi_ids[0], correct_this)
```

```
        else:
            if affi_id != None :
                print("Add author affiliation for author: ", art_author_id, 'with affi:', a
                new_affi_id = add_author_affiliation(db_name, art_aut_id, affi_id, affi_val
                #update cr_affis (assign author_affi_id)
                for cr_id in cr_affi_ids:
                    update_cr_aai(db_name, cr_id, new_affi_id)

def correct_multiline(db_name, cr_parser, cr_affis):
    # get a list of parsed affis with the ids of the corresponding cr_records
    parsed_affis = cr_parser.parse_and_map_multiline(cr_affis)
    print(parsed_affis)
    # all belong to same article author
    art_author_id = cr_affis[0][2]

    print ("verifying affiliations for article author", art_author_id)

    art_auth_affis = get_auth_affi_id_for_author(db_name, art_author_id)

    print ("Article author affiliations:", len(art_auth_affis), art_auth_affis )

    print ("Parsed article author affiliations:", len(parsed_affis) )

    if len(parsed_affis) > len(art_auth_affis):
        missing_author_affi = True

    for affi_idx, parsed_affi in enumerate(parsed_affis):
        affi_vals = parsed_affi[0]
        cr_affi_ids = parsed_affi[1]
        correct_this = 0
        if affi_idx < len(art_auth_affis):
            correct_this = art_auth_affis[affi_idx]

        affi_id = get_affiliation_id(db_name, affi_vals)
        if affi_id == None:
            parsed_no_blanks = {k:v for k,v in affi_vals.items() if v != ''}
            affi_id = get_close_affiliation_id(db_name, parsed_no_blanks)
        if correct_this != 0:
            # if the affiliation exists
            if affi_id != None:
                print('Update author_affiliation:', correct_this, 'with affi:', affi_vals )
                update_author_affiliation(db_name, correct_this, affi_id, affi_vals)
                for cr_id in cr_affi_ids:
                    update_cr_aai(db_name, cr_id, affi_id)
            else:
                print('Affi does not exist')
                print(affi_vals)

        else:
            if affi_id != None:
                print("Add author affiliation for author: ", art_author_id, 'with affi:', a
                new_affi_id = add_author_affiliation(db_name, art_aut_id, affi_id, affi_val
                #update cr_affis (assign author_affi_id)
                for cr_id in cr_affi_ids:
                    update_cr_aai(db_name, cr_id, new_affi_id)

def make_author_affiliation(art_aut_id, affi_values, addr_values):
    # get smallest unit
    smallest_unit = ""
    #id Institution> Faculty > School > Department > Work_group + address + Country
    if affi_values[4] != None and  len(affi_values[4]) > 0: #'work_group'
        smallest_unit = affi_values[4]
    elif affi_values[2] != None and len(affi_values[2]) > 0 and smallest_unit == "": #'depa
        smallest_unit = affi_values[2]
    elif affi_values[9] != None and  len(affi_values[9]) > 0 and smallest_unit == "": #'sch
        smallest_unit = affi_values[9]
    elif affi_values[3] != None and len(affi_values[3]) > 0 and smallest_unit == "": #'facu
        smallest_unit = affi_values[3]

    ret_art_auth_affi = {}
    ret_art_auth_affi['article_author_id'] = art_aut_id
    if len(smallest_unit) > 0:
        ret_art_auth_affi['name'] = smallest_unit + ", "+  affi_values[1] #'institution'
    else:
        ret_art_auth_affi['name'] = affi_values[1]
    ret_art_auth_affi['short_name'] = affi_values[1]
    add_01 = ""
    if affi_values[3] != None and affi_values[3]  != "" and affi_values[3] != smallest_unit
        add_01 = affi_values[3]
    if affi_values[9] != None and affi_values[9] != "" and affi_values[9] != smallest_unit:
        if add_01 != "":
                add_01 += ", "+ affi_values[9]
        else:
                add_01 += affi_values[9]
    if affi_values[2] != None and affi_values[2] != "" and affi_values[2] != smallest_unit:
        if add_01 != "":
                add_01 += ", "+ affi_values[2]
        else:
                add_01 += affi_values[2]
    if add_01 != "":
        ret_art_auth_affi['add_01'] = add_01
        ret_art_auth_affi['add_02'] = addr_values[1]
        ret_art_auth_affi['add_03'] = addr_values[2]
        ret_art_auth_affi['add_04'] = addr_values[3]
        ret_art_auth_affi['add_05'] = addr_values[4]
    else:
        ret_art_auth_affi['add_01'] = addr_values[1]
        ret_art_auth_affi['add_02'] = addr_values[2]
        ret_art_auth_affi['add_03'] = addr_values[3]
        ret_art_auth_affi['add_04'] = addr_values[4]

    ret_art_auth_affi['country'] = addr_values[5]
    ret_art_auth_affi['affiliation_id'] = affi_values[0]
    ret_art_auth_affi['created_at'] = datetime.today().strftime('%Y-%m-%d %H:%M:%S')
    ret_art_auth_affi['updated_at'] = ret_art_auth_affi['created_at']
    return ret_art_auth_affi

def build_address_row(affi, affi_vals):
    address_row = [0,None,None,None,None,None]
    if 'address' in affi_vals:
        address_row[1] = affi_vals['address']
    if 'country' in affi_vals:
        address_row[5] = affi_vals['country']
    else:
        address_row[5] = af
    return address_row

def add_author_affiliation(db_name, art_aut_id, affi_id, affi_values):
    print("Creating ", art_aut_id, affi_id, affi_values)
    db_conn = dbh.DataBaseAdapter(db_name)
    affiliation_row = list(db_conn.get_row("affiliations", affi_id))[0]
    address_row = build_address_row(affiliation_row, affi_values)
    print("Affiliation values", affiliation_row)
    print("Address values", address_row )
    new_auth_affi = make_author_affiliation(art_aut_id, affiliation_row, address_row)
    print('Adding:', new_auth_affi)
    new_aa_id = db_conn.put_values_table("author_affiliations", new_auth_affi.keys(), new_a
    return new_aa_id

def is_affi_ok(an_affi):
```

```
    affi_ok = True
    print ("Cheking affi:", an_affi)
    # has institution and institution is not blank
    if an_affi['institution'] == '' or an_affi['institution'] == None:
        print('Affiliation Error: Missing institution')
        affi_ok = False
    if an_affi['country'] == '' or an_affi['country'] == None:
        print('Affiliation Error: country missing')
        affi_ok = False

    if an_affi['sector'] == '' or an_affi['sector'] == None:
        sector_sel = 0
        while not sector_sel in [1,2,3]:
            print('Affiliation Error: sector missing')
            print("\n1. Academia\n2. Industry\n3. Research Facility",
                  "\nSelect sector: ")
            sector_sel = int(input())
        an_affi['sector'] = sector_sel

    return affi_ok


def add_new_affiliation(db_name, affi_values):
    affi_values['sector']=''

    if not is_affi_ok(affi_values):
        return 0;
    db_conn = dbh.DataBaseAdapter(db_name)
    add_update_time = datetime.today().strftime('%Y-%m-%d %H:%M:%S')
    affiliation_new = affi_values
    del affiliation_new['address']
    if 'address' in affiliation_new.keys():
        del affiliation_new['address']
    if 'num' in affiliation_new.keys():
        del affiliation_new['num']
    affiliation_new['created_at'] = add_update_time
    affiliation_new['updated_at'] = add_update_time
    affiliation_id = db_conn.put_values_table("affiliations", affiliation_new.keys(), affil
    return affiliation_id

def update_author_affiliation(db_name, aut_affi_id, affi_id, affi_values):
    print("Updating", aut_affi_id, affi_values)
    db_conn = dbh.DataBaseAdapter(db_name)
    affiliation_row = list(db_conn.get_row("affiliations", affi_id))[0]
    address_row = [0,None,None,None,None,None]
    if 'address' in affi_values:
        address_row[1] = affi_values['address']
    if 'country' in affi_values:
        address_row[5] = affi_values['country']
    else:
        address_row[5] = affiliation_row[5]

    auth_affi = make_author_affiliation(0, affiliation_row, address_row)
    update_time = datetime.today().strftime('%Y-%m-%d %H:%M:%S')

    print(auth_affi)
    for affi_col in auth_affi:
        if not affi_col in ["article_author_id", "created_at"]:
            new_value = auth_affi[affi_col]
            if isinstance(new_value, str):# new_value != None and not isinstance(new_value,
                if "'" in new_value: new_value = new_value.replace("'","''")
                if "’" in new_value: new_value = new_value.replace("’","''")
            print("updating aut_affi_id:", aut_affi_id, "column:", affi_col, "value:", new_
            db_conn.set_value_table('author_affiliations', aut_affi_id,  affi_col, new_valu
```

```
def update_cr_aai(db_name, cr_affi_id, auth_affi_id):
    db_conn = dbh.DataBaseAdapter(db_name)
    s_table = 'cr_affiliations'
    s_field = 'author_affiliation_id'
    db_conn = dbh.DataBaseAdapter(ukchapp_db)
    db_conn.set_value_table(s_table, cr_affi_id,  s_field , auth_affi_id)


########################################################################
# VERIFY ARTICLE AUTHOR AFFILITIONS VS CR_AFFILIATIONS
# 0 Verify integrity of affiliations
# 1 Get list of article_author_ids from CR_affi
# 2 For each article_author_id:
#   1 Get CR_affi lines
#   2 verify CR_affi lines
#     1 check if
#        a) one affiliation per cr_affi
#           parse each single affiliation
#           check if assigned affiliation is OK (assigned ID matches calculated ID)
#        b) multiple lines form an affiliation (2+)
#           parse each multi-line affiliation
#           check if assigned affiliation is OK

affi_parser = get_parser(ukchapp_db)

list_art_aut_ids = get_cr_affis_article_author_ids(ukchapp_db)

already_ok = open_ok_list('ok_affi_list.txt')

for art_aut_id in list_art_aut_ids:
    if not art_aut_id in already_ok:
        print ('Article Author: ', art_aut_id)
        cr_lines = get_cr_lines_for_article_author_ids(ukchapp_db, art_aut_id)
        print('{0:*^80}'.format('CR Affilitations found:'), "\n", cr_lines)
        all_one_liners = True
        print('{0:*^80}'.format('Check if CR lines are one liners:'))
        for a_cr_line in cr_lines:
            one_line_affi = is_one_line_affi(affi_parser, a_cr_line[1])
            print( a_cr_line[1], one_line_affi)
            if not one_line_affi:
                all_one_liners = False
        if all_one_liners:
            assigned_ok = False
            print('{0:*^80}'.format('verify one liners'))
            for a_cr_line in cr_lines:
                assigned_ok = check_assigned_affi_ol(ukchapp_db, affi_parser, a_cr_line)
                print(assigned_ok)
                if not assigned_ok:
                    print("Problems with ", a_cr_line[0])
                    break
            if not assigned_ok:
                correct_oneline(ukchapp_db, affi_parser, cr_lines)
                #break
            else:
                already_ok.append(art_aut_id)
        else:
            print('verify multiline affi')
            assigned_ok = check_assigned_affi_ml(ukchapp_db, affi_parser, cr_lines, art_aut
            if not assigned_ok:
                print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@")
                print("Problems with:\n", cr_lines[0][2], art_aut_id)
                correct_multiline(ukchapp_db, affi_parser, cr_lines)
                #break
            else:
```

```
                already_ok.append(art_aut_id)

print("OK:", len(already_ok))
print(already_ok)
save_ok_list(already_ok, 'ok_affi_list.txt')
```

In [ ]:
```
a_cr_line
```

In [ ]:
```
already_ok.sort()
print("OK:", len(already_ok))
print(already_ok)
save_ok_list(already_ok, 'ok_affi_list.txt')
```

In [ ]:
```
cr_lines = get_cr_lines_for_article_author_ids(ukchapp_db, 207)
print(cr_lines)
parsed_affis = [[{'institution': 'University College London', 'school': '', 'department': '
print(parsed_affis[0][0])
del parsed_affis[0][0]['address']
print(parsed_affis[0][0])
```

Check that pdf files exist

Use the data on the articles table to verify if file are stored in the corresponding folder We also check that the files in the folder are all accounted for (have a corersponding record)

In [ ]:
```
if current_step == 2:
    # get publication data from the ukch app
    app_pubs = pr_fns.get_pub_data(ukchapp_db)

    dups = []
    for idx, a_pub in enumerate(tqdm_notebook(app_pubs)):
        pub_id = a_pub[0]
        pub_title = a_pub[1]
        pub_doi = a_pub[2]
        pub_url = a_pub[3]
        for i_indx in range(idx,len(app_pubs)):
            if not (pub_doi is None) and pub_doi.strip().lower() ==  app_pubs[i_indx][2]:
                print(pub_doi, "duplicated at:", i_indx)
```

In [ ]:
```
ukchapp_db
```

In [ ]:
```
if current_step == 2:
    for infile in tqdm_notebook(Path("pdf_files").glob('*.pdf')):
        file_found = False
        for a_pub in app_pubs:
            if infile.name == a_pub[4]:
                file_found = True
                break
        if not file_found:
            print("Not in DB:", infile.name, "DB Name", a_pub[4])
```

Get missing pdfs

In [ ]:
```
# use regular expression to check if a given string
# is a valid DOI, using pattern from CR
def valid_doi(cr_doi):
    # CR DOIS: https://www.crossref.org/blog/dois-and-matching-regular-expressions/
    # CR DOIs re1
    # /^10.\d{4,9}/[-._;()/:A-Z0-9]+$/i
    if cr_doi == None:
        return False
    cr_re_01 = '^10.\d{4,9}/[-._;()/:A-Z0-9]+'
    compare = re.match(cr_re_01, cr_doi, re.IGNORECASE)
    if compare != None and cr_doi == compare.group():
        return True
    else:
        return False
# get publication data from the ukch app
db_pubs = pr_fns.get_pub_data(ukchapp_db)

if current_step == 2:
    for a_pub in tqdm_notebook(db_pubs):
        if a_pub[0] > 616:
            pub_id = a_pub[0]
            pub_title = a_pub[1]
            pub_doi = a_pub[2]
            pub_url = a_pub[3]
            pub_pdf = a_pub[4]
            pub_html = a_pub[5]
            if pub_pdf == None:
                not_in_url = True
                print("ID: ", pub_id, "Publication: ",pub_title,
                      "\n\tDOI: ", pub_doi, " URL: ", pub_url)
                if "pdf" in pub_url:
                    print ("\tTry to get the pdf from URL: ", pub_url)
                    try:
                        response = requests.get(pub_url)
                        content_type = response.headers['content-type']
                        if not 'text' in content_type:
                            #print(response.headers)
                            cd= response.headers['content-disposition']
                            #print(cd)
                            fname = re.findall("filename=(.+)", cd)[0]
                            #print(fname)
                            if not Path('pdf_files/' + pdf_file).is_file():
                                with open('pdf_files/'+ fname +'.pdf', 'wb') as f:
                                    f.write(response.content)
                            else:
                                set_pdf_file_value(pdf_file, pub_id, ukchapp_db)
                            not_in_url = False
                    except:
                        print("ID: ", pub_id, "\nPublication: ",pub_title,
                              "\nDOI: ", pub_doi, "\nDOI: ", pub_url)
                if not_in_url:
                    print("\tTry to see if json file has link to pdf: ")
                    if valid_doi(pub_doi):
                        crjd, doi_file = pr_fns.get_cr_json_object(pub_doi)
                        got_pdf = False
                        if "link" in crjd.keys():
                            for a_link in crjd["link"]:
                                if "\tURL" in a_link.keys() and ("pdf" in a_link["URL"] or
                                    cr_url = a_link["URL"]
                                    #print("URL: ", cr_url)
                                    pdf_file = get_pdf_from_url(cr_url)
                                    # if the name corresponds to a existing file, assign va
                                    if Path('pdf_files/' + pdf_file).is_file():
                                        print("\tFile name:", pdf_file)
```

```
                    set_pdf_file_value(pdf_file, pub_id, ukchapp_db)
                    got_pdf = True
                else:
                    print("\tcould not get file from", cr_url)
    else:
        print("\tno links in json", pub_doi)
if not got_pdf and "elsevier" in pub_url:
    print("\tTrying elsevier doi:" )
    pdf_file = pr_fns.get_elsevier_pdf(pub_doi)
    if Path('pdf_files/' + pdf_file).is_file():
        print("\tFile name:", pdf_file)
        pr_fns.set_pdf_file_value(pdf_file, pub_id, ukchapp_db)
        got_pdf = True
elif not got_pdf and "wiley" in pub_url:
    print("\tTrying wiley doi:" )
    pdf_file = pr_fns.get_wiley_pdf(pub_doi)
    if Path('pdf_files/' + pdf_file).is_file():
        print("\tFile name:", pdf_file)
        pr_fns.set_pdf_file_value(pdf_file, pub_id, ukchapp_db)
        got_pdf = True
elif not got_pdf and "pubs.acs" in pub_url:
    print("\tTrying acs doi:" )
    pdf_file = pr_fns.get_acs_pdf(pub_doi)
    if Path('pdf_files/' + pdf_file).is_file():
        print("\tFile name:", pdf_file)
        pr_fns.set_pdf_file_value(pdf_file, pub_id, ukchapp_db)
        got_pdf = True
if not got_pdf:
    print("\tTry doi:  https://doi.org/" + pub_doi)
```

Use pdfminer to get metadata from pdf file

In [ ]:

```
import pdfminer
from pdfminer import high_level as pdfmnr_hl

# functions for PDFminer

def get_pdf_text(pdf_file):
    return pdfmnr_hl.extract_text(pdf_file)

# get the paragraph fragments with references to data
def get_ref_sentences(pdf_text):
    sentences = pdf_text.split("\n")
    groups=[]
    for sentence in sentences:
        if pr_fns.is_data_stmt(sentence.lower()):
            idx = sentences.index(sentence)
            groups.append([idx-1,idx,idx+1])
    reduced_groups = []
    for group in groups:
        idx_group = groups.index(group)
        if groups.index(group) > 0:
            set_g = set(group)
            # make the array before current a set
            set_bg = set(groups[idx_group - 1])
            # make the array after current a set
            set_ag = set()
            if idx_group + 1 < len(groups):
                set_ag = set(groups[idx_group + 1])
            if len(set_bg.intersection(set_g)) > 0:
                ordered_union = list(set_bg.union(set_g))
                ordered_union.sort()
                reduced_groups.append(ordered_union)
            if len(set_ag.intersection(set_g)) > 0:
                ordered_union = list(set_ag.union(set_g))
                ordered_union.sort()
                reduced_groups.append(ordered_union)
            if len(reduced_groups) > 0:
                is_in_rg = False
                for a_rg in reduced_groups:
                    if set_g.issubset(a_rg):
                        is_in_rg = True
                        break
                if not is_in_rg:
                    reduced_groups.append(list(set_g))
    return_group = []
    for sentence_group in reduced_groups:
        full_sentence = ""
        for single_sentence in sentence_group:
            full_sentence += sentences[single_sentence].strip()
        return_group.append(full_sentence)
    return return_group

# get the paragraph fragments with references to data
def get_all_data_sentences(pdf_text):
    sentences = pdf_text.split("\n")
    groups=[]
    for sentence in sentences:
        if 'data' in sentence.lower() or 'inform' in sentence.lower():
            idx = sentences.index(sentence)
            groups.append([idx-1, idx, idx+1])
    reduced_groups = []
    for group in groups:
        idx_group = groups.index(group)
        if groups.index(group) > 0:
            set_g = set(group)
```

```
            # make the array before current a set
            set_bg = set(groups[idx_group - 1])
            # make the array after current a set
            set_ag = set()
            if idx_group + 1 < len(groups):
                set_ag = set(groups[idx_group + 1])
            if len(set_bg.intersection(set_g)) > 0:
                ordered_union = list(set_bg.union(set_g))
                ordered_union.sort()
                reduced_groups.append(ordered_union)
            if len(set_ag.intersection(set_g)) > 0:
                ordered_union = list(set_ag.union(set_g))
                ordered_union.sort()
                reduced_groups.append(ordered_union)
            if len(reduced_groups) > 0:
                is_in_rg = False
                for a_rg in reduced_groups:
                    if set_g.issubset(a_rg):
                        is_in_rg = True
                        break
                if not is_in_rg:
                    reduced_groups.append(list(set_g))
    return_group = []
    for sentence_group in reduced_groups:
        full_sentence = ""
        for single_sentence in sentence_group:
            full_sentence += sentences[single_sentence].strip()
        if not full_sentence in return_group:
            return_group.append(full_sentence)
    return return_group

# get the http strings from references to data
def get_http_ref(sentence):
    http_frag = ""
    if 'http' in sentence.lower():
        idx_http = sentence.lower().index('http')
        http_frag = sentence[idx_http:]
        space_in_ref = True
        while " " in http_frag:
            space_idx = http_frag.rfind(" ")
            http_frag = http_frag[:space_idx]
        if(http_frag[-1:]=="."):
            http_frag = http_frag[:-1]
    return http_frag
```

In [ ]:
```
if current_step == 2:
    # get publication data from the ukch app
    db_pubs = pr_fns.get_pub_data(ukchapp_db)

    # get the list of dois already mined for data
    input_file = 'pub_data_add202012.csv'
    id_field = 'num'
    processed, headings = csvh.get_csv_data(input_file, id_field)
    for id_num in processed:
        current_title = processed[id_num]['doi']
    processed[1]['num']

    processed_dois = []
    for entry in processed:
        if not processed[entry]['doi'] in processed_dois:
            processed_dois.append( processed[entry]['doi'])

    data_records = {}
    data_mentions = {}
    ref_count = mention_count = 0
```

```
    for a_pub in tqdm_notebook(db_pubs):
        data_refs = []
        data_sents = []
        if a_pub[0] > 616:
            pub_id = a_pub[0]
            pub_title = a_pub[1]
            pub_doi = a_pub[2]
            pub_url = a_pub[3]
```

In [ ]:
```
#if len(data_records) > 0:
#    csvh.write_csv_data(data_records, 'pdf_data.csv')
if current_step == 2:
    if len(data_mentions) > 0:
        csvh.write_csv_data(data_mentions, 'pdf_mentions202110.csv')
                print(
            else:
                pdf_file = "pdf_files/" + pub_pdf
                if not Path(pdf_file).is_file():
```

In [ ]:
```
from IPython.display import clear_output

if current_step == 3:
    print(ukchapp_db)
    print(len(app_pubs))
    # Open results file
    data_mentions, dm_headers = csvh.get_csv_data('pdf_mentions202110.csv')
    print(dm_headers)
    art_id = ''
    for dm in data_mentions:
        if data_mentions[dm]['action']=='':
            clear_output()
            print ("*****************************************")
            print ("Article id  :", data_mentions[dm]['id'])
            print ("DOI         :", data_mentions[dm]['doi'])
            print ("Type        :", data_mentions[dm]['type'], '\tLine:', dm)
            print ("Description :\n\t", data_mentions[dm]['desc'])
            print ("data_url :", data_mentions[dm]['data_url'])
            print ("*****************************************")
            decide_action = False
            while not decide_action:
                print('Action:')
                print('\ta) review')
                print('\tb) none')
                print('\tSelect a or b:')
                lts = input()
                if lts == "a":
                    data_mentions[dm]['action'] = 'review'
                    decide_action = True
                elif lts == "b":
                    data_mentions[dm]['action'] = 'none'
                    decide_action = True
        art_id = data_mentions[dm]['id']
        if dm > 1700:
            break
    if len(data_mentions) > 0:
       csvh.write_csv_data(data_mentions, 'pdf_mentions202110.csv')
```

In [ ]:
```
# clear the output after each loop cycle
from IPython.display import clear_output

# display editable spreadsheet
import ipysheet


# show gds parameters in a spreadsheet on jupyter
def show_gds(gds_group):
    gds_list = gds_to_list(gds_group)
    #print(gds_list)
    #add 10 more rows in case we need more parameters
    for i in range(10):
        gds_list.append([(len(gds_list)-1)+1,None,None,None,None])
    a_sheet = ipysheet.sheet(rows=len(gds_list), columns=len(gds_list[0]))
    ipysheet.cell_range(gds_list)
    display(a_sheet)
    return a_sheet

if current_step == 3:
    print(ukchapp_db)
    print(len(app_pubs))
    # Open results file
    data_mentions, dm_headers = csvh.get_csv_data('pdf_mentions202110.csv')
    print(dm_headers)
    art_id = ''
    terminate = False
    additional_rows = {}
    for dm in data_mentions:
        if data_mentions[dm]['action']=='review':
            clear_output()
            print ("*****************************************")
            print ("Article id  :", data_mentions[dm]['id'])
            print ("DOI         :", data_mentions[dm]['doi'])
            print ("Type        :", data_mentions[dm]['type'], '\tLine:', dm)
            print ("Description :\n\t", data_mentions[dm]['desc'])
            print ("data_url :", data_mentions[dm]['data_url'])
            print ("*****************************************")
            decide_action = False
            while not decide_action:
                print('Action:')
                print('\ta) review: https://doi.org/'+data_mentions[dm]['doi'])
                print('\ts) add new row')
                print('\td) next')
                print('\tf) terminate')
                print('\tSelect a, s, d, f:')
                lts = input()
                if lts == "a":
                    data_mentions[dm]['action'] = 'reviewed'
                    print ('https://doi.org/'+data_mentions[dm]['doi'])
                    print ('link:',data_mentions[dm]['link'])
                    add_this = input()
                    data_mentions[dm]['link'] = add_this
                    print ('issue:',data_mentions[dm]['issue'])
                    add_this = input()
                    data_mentions[dm]['issue'] = add_this
                    print ('name:',data_mentions[dm]['name'])
                    add_this = input()
                    data_mentions[dm]['name'] = add_this
                    print ('file:',data_mentions[dm]['file'])
                    add_this = input()
                    data_mentions[dm]['file'] = add_this
                if lts == "s":
                    #add a new row
                    new_idx = len(data_mentions) + len(additional_rows) + 1
                    additional_rows[new_idx] = data_mentions[dm]
                    print ('link:',additional_rows[new_idx]['link'])
                    add_this = input()
                    additional_rows[new_idx]['link'] = add_this
                    print ('issue:',additional_rows[new_idx]['issue'])
                    add_this = input()
                    additional_rows[new_idx]['issue'] = add_this
                    print ('name:',additional_rows[new_idx]['name'])
                    add_this = input()
                    additional_rows[new_idx]['name'] = add_this
                    print ('file:',additional_rows[new_idx]['file'])
                    add_this = input()
                    additional_rows[new_idx]['file'] = add_this
                elif lts == "d":
                    if data_mentions[dm]['action'] != 'reviewed':
                        data_mentions[dm]['action'] = 'none'
                    decide_action = True
                elif lts == 'f':
                    decide_action = True
                    terminate = True
            art__id = data_mentions[dm]['id']
            if dm > 1700 or terminate:
                break
        if len(additional_rows)> 0 :
            for nr in additional_rows:
                for a_field in dm_headers:
                    data_mentions[nr][a_field] = additional_rows[nr][a_field]
        if len(data_mentions) > 0:
            csvh.write_csv_data(data_mentions, 'pdf_mentions202110.csv')
```

In [ ]:
```
filter_mentions = {}
for dm in data_mentions:
    if data_mentions[dm]['action'] in ['add', 'reviewed']:
        filter_mentions[dm]={}
        for a_field in dm_headers:
            filter_mentions[dm][a_field] = data_mentions[dm][a_field]
print('filtered mentions:', len(filter_mentions))
```

In [ ]:
```
new_do_id_list =[]
for fm in filter_mentions:
    art_id = int(filter_mentions[fm]["id"])
    if not art_id in new_do_id_list:
        new_do_id_list.append(art_id)

# currend app DB
ukchapp_db = "db_files/app_db20211005.sqlite3"

no_data_pubs = pr_fns.get_pub_app_no_data(ukchapp_db)
#print(len(ids_w_data))
print(len(no_data_pubs))
print(new_do_id_list, len(new_do_id_list))
filter_mentions


int_idx = 0
revised_list = {}
if Path("./html_revised202111.csv").is_file():
    revised_list, rl_headers = csvh.get_csv_data('html_revised202111.csv')
    int_idx = len(revised_list)

already_revised =[]
for fm in revised_list:
    art_id = int(revised_list[fm]["id"])
    if not art_id in already_revised:
        already_revised.append(art_id)

for ndp in no_data_pubs:
    if not ndp[0] in new_do_id_list and ndp[0] > 616 and not ndp[0] in already_revised:
        int_idx += 1
        pub_id = ndp[0]
        pub_title = ndp[1]
        pub_doi = ndp[2]
        pub_url = ndp[3]
        data_record = {'id':pub_id, 'doi':pub_doi, 'title':pub_title}
        print ('id',pub_id, '\n', pub_title)
        decide_action = False
        terminate = False
        while not decide_action:
            print('Action:')
            print(pub_url)
            print("https://doi.org/"+pub_doi)
            print('\ta) no data' )
            print('\ts) review')
            print('\td) next')
            print('\tf) terminate')
            print('\tSelect a, s, d, f:')
            lts = input()
            if lts == "a":
                data_record['action'] = 'no data'
                data_record['issue'] = "no data availability or supplementary data mentione
                revised_list[int_idx] = data_record
                decide_action = True
            if lts == "s":
                data_record['action'] = 'review'
                print ('issue:',data_mentions[dm]['issue'])
                add_this = input()
                data_record['issue'] = add_this
                revised_list[int_idx] = data_record
                decide_action = True
            if lts == "d":
                decide_action = True
            elif lts == 'f':
                decide_action = True
                terminate = True
        if terminate:
            break

if len(revised_list) > 0:
    csvh.write_csv_data(revised_list, 'html_revised202111.csv')
```

In [ ]:
```
if len(revised_list) > 0:
    csvh.write_csv_data(revised_list, 'html_revised202111.csv')
revised_list
```

In [ ]:
```
# functions for ChemDataExtractor
# not used for mining data references (suplementary/raw) or to get pdf metadata
from chemdataextractor import Document

# A function for getting a list of files from the directory
# This will be modified to get the list from a csv file
def get_files_list (source_dir):
    i_counter = 0
    files_list = []
    for filepath in sorted(source_dir.glob('*.pdf')):
        i_counter += 1
        files_list.append(filepath)
    return files_list

def cde_read_pdfs(a_file):
    pdf_f = open(a_file, 'rb')
    doc = Document.from_file(pdf_f)
    return doc

def find_doi(element_text):
    cr_re_01 = '10.\d{4,9}/[-._;()/:A-Z0-9]+'
    compare = re.search(cr_re_01, element_text, re.IGNORECASE)
    if compare != None:
        return compare.group()
    return ""

def get_db_id(doi_value, db_name = "app_db.sqlite3"):
    db_conn = dbh.DataBaseAdapter(db_name)
    table = 'articles'
    id_val = db_conn.get_value(table, "id", "doi", doi_value)
    db_conn.close()
    if id_val != None:
        return id_val[0]
    else:
        return 0

def get_db_title(doi_value, db_name = "app_db.sqlite3"):
    db_conn = dbh.DataBaseAdapter(db_name)
    table = 'articles'
    id_val = db_conn.get_value(table, "title", "doi", doi_value)
    db_conn.close()
    if id_val != None:
        return id_val[0]
    else:
        return 0

def get_close_dois(str_name, db_name = "prev_search.sqlite3"):
    db_conn = dbh.DataBaseAdapter(db_name)
    search_in = 'articles'
    fields_required = "id, doi, title, pdf_file"
    filter_str = "doi like '%"+str_name+"%';"

    db_titles = db_conn.get_values(search_in, fields_required, filter_str)
    db_conn.close()
    return db_titles
```

Get the name of the current app db file:

In [ ]:
```
# app db file with path: db_files/app_db.sqlite3
ukchapp_db = "db_files/app_db2.sqlite3"
while not Path(ukchapp_db).is_file():
    print('Please enter the name of app db file:')
    ukchapp_db = input()
ukchapp_db
```

In [ ]:
```
# get names and links for references in data mentions
data_mentions, dm_fields = csvh.get_csv_data('pdf_mentions_filtered_02.csv', 'num')

for dm in data_mentions:
    print("https://doi.org/" + data_mentions[dm]['doi'])
    ref_name = data_mentions[dm]['ref_name']
    while ref_name == "":
        print('Please enter the name of data object:')
        ref_name = input()
    ref_link = data_mentions[dm]['ref_link']
    while ref_link == "":
        print('Please enter the data object link:')
        ref_link = input()
    data_mentions[dm]['ref_name'] = ref_name
    data_mentions[dm]['ref_link'] = ref_link
```

In [ ]:
```
len(data_records)
```

In [ ]:
```
data_mentions
```

In [ ]:
```
from inspect import getmembers, isfunction
```

In [ ]:
```
help(pdfminer.high_level)
```

In [ ]:
```
!jupyter --version
```

In [ ]:
```
from platform import python_version
python_version()
```

In [ ]:
```
db_conn = dbh.DataBaseAdapter(ukchapp_db)
table_name = "Articles"
column_name = "pdf_file"
db_conn.connection.execute("ALTER TABLE " + table_name + " DROP COLUMN " + column_name + ";
```

In [ ]:
```
```