IEc

CHARACTERIZING ORGANOHALOGEN FLAME RETARDANT (OFR) CHEMISTRIES, SOURCES, AND USES IN UNITED STATES AND INTERNATIONAL MARKETS

Attachment A: Standard Operating Procedure for Producing Data Source Outputs

Contract No. 47QRAA20D0044

Order No. 61320621F0021

Final Report | March 2022

Prepared for:

U.S. Consumer Product Safety Commission (CPSC)
Directorate for Health Sciences
5 Research Place
Room 165-01
Rockville, MD 20850

Prepared by:
Dr. Ann Jones, Jennifer Baxter, Dr. Rita Cabral, and Eden Blutstein

Industrial Economics, Incorporated (IEc)
2067 Massachusetts Avenue
Cambridge, MA 02140
617/354-0074

&

Jeff Cantin, Daryl Hudson, Kurt Rindfusz, Anna Dimling, Owen Stokes-Cawley, Abby Burton, Taylor Carlough, and George Wieber

Eastern Research Group, Incorporated (ERG)
110 Hartwell Avenue
Lexington, MA 02421
781/674-7200

INDUSTRIAL ECONOMICS, INCORPORATED

This page intentionally left blank.

STANDARD OPERATING PROCEDURE FOR PRODUCING DATA SOURCE OUTPUTS

This Standard Operating Procedure (SOP) provides the code run in the Spyder integrated development environment (IDE) utilizing Python language (Anaconda Navigator version 2.1.0, Spyder version 5.1.5). Data sources were either obtained through personal communication with United States Environmental Protection Agency (U.S. EPA) and Consumer Product Safety Commission (CPSC) staff or downloaded directly from online data sources. In general, these data sources were in Microsoft Excel format (either as comma separated values [CSV] or Excel workbook [XLSX] files). However, PubChem supports Application Programming Interface (API) queries, which allows two applications to interface with one another. This feature allows queries to be run on PubChem servers, with the results returned to the user in Spyder.

One important aspect of the various queries is the identifier used to recall data for a particular chemical. For example, the "native" identifier used in PubChem is a compound identifier (cid) while CompTox uses a DSSTox substance identifier (DTXSID), and other sources may simply use the Chemical Abstracts Service (CAS) number. As a result, the query results may be somewhat different from each data source. Sometimes this occurs because of the way naming systems generate identifiers for compounds that have mixed isomers. For example, one cid (45472) could be associated with multiple CAS numbers (63936-56-1 and 63387-28-0).[1] This data process defaults to using the "native" identifier for each data source and handles the returned query results as they are.

The Python code for querying PubChem directly is provided in Section A.1. The Python code used for importing all the data and combining them with the list of relevant Organohalogen Flame Retardant (OFR) chemicals identified by the Consumer Product Safety Commission (CPSC) is provided in Section A.2. The narrative description of this process as well as the data and sources themselves are described in Chapter 2 within the main body of this report. The set of database files that are generated by this process accompanies the report.

A.1 PYTHON CODE FOR QUERYING PUBCHEM AND RELATED CAVEATS

This code queries PubChem for chemical information associated with the list of OFR cids (Attachment 1). It interfaces directly with PubChem servers for standard physical and chemical information, but it also scrapes the JSON (JavaScript Object Notation) for specific parameters (e.g., boiling point, solubility). The data that is returned by the latter process is not always in a standard notation, and presumably reflects how the value was reported in the primary source. Standardizing these values could be a logical next step during further data processing.

Another caveat relates to the patent information retrieved from PubChem. Querying patent-related information (e.g., abstract, title, date) requires two steps. First, a list of patent numbers is retrieved for each cid; these patent numbers are stored as a list on the compound's PubChem page. Second, each patent

[1] https://pubchem.ncbi.nlm.nih.gov/compound/45472#section=CAS

number is then used as a component of a website URL (Uniform Resource Locator) to access the patent's PubChem page, which contains the detailed patent information. In some cases, the format of the patent number retrieved from PubChem does not match the format used in the URL, which prevents the detailed information from being queried for a given patent (using the patent page's JSON, as above). The code attempts to correct for some of the more obvious formatting issues (e.g., adding a hyphen after the country prefix or before the patent suffix [changing "US4923973A" to "US-4923973-A"], and modifying the formatting of certain U.S. patent numbers [changing "US-20080132638-A1" to "US-2008132638-A1"]), but these corrections may not be exhaustive. After applying these corrections, approximately 12 percent of OFR-related patents are not able to be matched to a PubChem patent page for retrieval of detailed patent information. For these reasons, the detailed patent information is incomplete. It may, therefore, be used as one line of evidence for the potential uses of OFR chemicals but is not exhaustive and, thus, may not be appropriate as the only line of evidence.

```
# -*- coding: utf-8 -*-
"""
Custom functions to query PubChem using pubchempy package
and general web scraping functions.

Created on Mon May 17 14:45:13 2021
@author: BCox (IEc)

User guide:
    [1] Change file path in setup cell to specify path for inputs and outputs
        Inputs should be in a folder called 'Input', and outputs will be saved
        to a subfolder called 'Output', both nested under main folder specified
    [2] Specify OFR universe file
    [3] Specify additional data fields
    [4] Specify whether code should generate CIDs based on OFR universe file
    [5] Specify whether code should query PubChem website
    [5] Run code sections (note: PubChem queries can take a very long time;
                                 querying all patents required 2+ days
                                 of unmonitored run time)

"""
```

```
#%% ============================================================================
#   SETUP AND PARAMETERS
#   ============================================================================

#%% Setup & key parameters
import pandas as pd
import numpy as np
import requests
import pubchempy as pcp
import time
import os
from collections import OrderedDict

# Set directory
os.chdir(r'C:/users/bcox/desktop/IEc Projects/PubChem/')

# Check that input and output folders exist; if not, create them
for p in ['Input','Output']:
    if not os.path.exists(p):
        os.makedirs(p)

# Determine whether to query PubChem for CIDs for OFR CAS numbers
pull_ofr_cids = False
# If yes, specify 'raw' OFR universe file
ofr_file_raw = 'Input/OFR_universe_12072021.csv'
# If no, specify OFR file with CIDs already added
    #ofr_file_with_cids = "Output/OFR_universe_cid_10082021.xlsx"
ofr_file_with_cids = "Output/OFR_universe_cid_12082021.xlsx"
ofr_file_with_cids = "Output/OFR_universe_cid_03042022.xlsx"

# Determine whether to query PubChem for various data or use existing results
```

```
query_pubchem = False

query_patents = False

read_patent_pickles = True


# Specify fields to be queried

additional_data_fields = ['Solubility',

                          'Boiling Point',

                          'Vapor Pressure',

                          'LogP',

                          'Flash-Point',

                          'Henrys-Law-Constant',

                          'LogKoa']

sections_to_query = ['Uses',

                     'Identification',

                     'Environmental-Fate-Exposure-Summary',

                     'Artificial-Pollution-Sources']


#%% ============================================================================

#   DEFINE FUNCTIONS

#   ============================================================================


#%% Define functions


def combine_pubchem_dfs(chemical_list=None, namespace=None):

    ''' Creates dataframe of compound records as retrieved

        by PubChemPy for a list of chemicals. '''

    if type(chemical_list) != list:

        raise Exception('Must specify list format input')

    if namespace == None:

        raise Exception("Must specify namespace input: 'cid', 'name', etc.")
```

```
    dfs = []
    for c in chemical_list:
        try:
            df = pcp.get_compounds(c, namespace=f'{namespace}', as_dataframe=True)
            dfs.append(df)
        except:
            print(f'pcp.get_compounds() fails for \'{namespace}\' \'{c}\'')

    # Combine into single dataframe
    df = pd.concat(dfs, axis=0)

    df.reset_index(drop=False, inplace=True)
    return df


def additional_data(cid=None, url_name=None, column_name=None, print_status=False):
    ''' Create dataframe of additional information for a single
        chemical compound (specified by CID) from PubChem.
        Specify URL name to focus search, and column name to specify resulting
        column name.
        Note: These data fields must be pulled using CID rather than CAS/etc. '''

    if (cid==None) | (url_name==None) | (column_name==None):
        raise Exception('Must specify parameters')

    if print_status==True:
        print(f'... retrieving {url_name} for CID {cid}')

    try:
        r =
requests.get(f'https://pubchem.ncbi.nlm.nih.gov/rest/pug_view/data/compound/%s/JSON/?heading={
url_name}' % cid)
```

```
        focus_json =
r.json()['Record']['Section'][0]['Section'][0]['Section'][0]['Information']


        ref_nums = [i['ReferenceNumber'] for i in focus_json]
        # Add detailed reference if available; otherwise np.nan
        ref_detail = []
        for i in focus_json:
            try:
                ref_detail.append(i['Reference'][0])
            except:
                ref_detail.append(np.nan)
        # Create lists of values
            # Note: Need error-catching for solubility values because some are
            # stored as 'StringWithMarkup' and others are stored as
            # 'Number' and 'Unit' combinations
        values = []
        for i in focus_json:
            if list(i['Value'].keys())[0] == 'StringWithMarkup':
                values.append(i['Value']['StringWithMarkup'][0]['String'])
            elif (list(i['Value'].keys())[0] == 'Number') & (list(i['Value'].keys())[1] ==
'Unit'):
                values.append(f"{i['Value']['Number'][0]} {i['Value']['Unit']}")


        # Create data frame for compound
        df = (pd.DataFrame(data={f'{column_name}': values,
                                 'ReferenceNumber': ref_nums,
                                 'Reference': ref_detail})
              .assign(cid=cid)) # Assign CID to allow merging


    # If no record is found in JSON, create dataframe with CID and np.nan
        # Note: Cannot simply 'pass' out of this; function returns None,
        # which causes problems in combining dataframes below,
```

```
    except:
          df = (pd.DataFrame(data={f'{column_name}': [np.nan],
                                   'ReferenceNumber': [np.nan]})
                .reset_index(drop=True)
                .assign(cid=cid))

    # Drop rows with no data
    df.dropna(subset=[f'{column_name}', 'ReferenceNumber'], axis=0, inplace=True)
    # Move cid column to first position
    df.insert(0, 'cid', df.pop('cid'))
    return df

def combine_additional_data(cid_list, url_name=None, column_name=None, print_status=False):
    ''' Compile single dataframe of additional PubChem information
        for a list of CIDs. Specify URL name and compound name to pass to
        underlying single-chemical function. '''
    if type(cid_list) != list:
        raise Exception('Must specify list format input')

    df = pd.concat([additional_data(c,
                                    url_name=url_name,
                                    column_name=column_name,
                                    print_status=print_status)
                    for c in cid_list],
                    axis=0).reset_index(drop=True)
    return df

def combine_section_check(cid_list=None, section_list=None, print_status=False):
    ''' Check if CID has specified section(s). If yes, return the URl. '''

    if (cid_list==None):
```

```
        raise Exception('Must specify parameters')

    dfs = []
    for cid in cid_list:
        data_by_cid = {}
        for section in section_list:
            if print_status==True:
                print(f'... retrieving {section} for CID {cid}')

            r =
requests.get(f'https://pubchem.ncbi.nlm.nih.gov/rest/pug_view/data/compound/{cid}/JSON/?headin
g={section}').json()

            if list(r.keys())[0] == 'Fault':
                data_by_cid[section] = np.nan
            else:
                data_by_cid[section] =
f'https://pubchem.ncbi.nlm.nih.gov/compound/{cid}#section={section}'

        # after looping through all sections for CID, append a df to list of dfs
        dfs.append(pd.DataFrame(data_by_cid, index=[0]).assign(cid=cid))

    # Concat and move cid column to first
    df = pd.concat(dfs, axis=0)
    df.insert(0, 'cid', df.pop('cid'))

    return df

def combine_synonyms(chemical_list=None, namespace=None):
    ''' Creates dataframe of compound synonyms as retrieved
        by PubChemPy for a list of chemicals. '''
    if type(chemical_list) != list:
```

```
        raise Exception('Must specify list format input')
    if namespace == None:
        raise Exception("Must specify namespace input: 'cid', 'name', etc.")

    dfs = []
    for c in chemical_list:
        try:
            df = pd.DataFrame(pcp.get_synonyms(c, namespace=f'{namespace}',
as_dataframe=True))
            dfs.append(df)
        except:
            print(f'pcp.get_synonyms() fails for \'{namespace}\' \'{c}\'')

    # Combine into single dataframe
    df = pd.concat(dfs, axis=0)

    df.reset_index(drop=True, inplace=True)
    df.rename({'CID':'cid'}, axis='columns', inplace=True)
    return df

def combine_references(cid_list=None):
    ''' Compile full reference list for list of compounds. '''
    if type(cid_list) != list:
        raise Exception('Must specify list format input')

    dfs = []
    for c in cid_list:
        try:
            df =
(pd.DataFrame(requests.get('https://pubchem.ncbi.nlm.nih.gov/rest/pug_view/data/compound/%s/JS
ON/' % c)
                                    .json()['Record']['Reference'])
```

```
                                        .assign(cid=c))
            dfs.append(df)
        except:
            print(f'No references for {c}')

    df = pd.concat(dfs, axis=0)
    df = df.sort_values(['cid','ReferenceNumber']).reset_index(drop=True)
    # Move cid column to first position
    df.insert(0, 'cid', df.pop('cid'))
    return df

def export_dfs(file, df_dict, **kwargs):
    ''' Export dictionary of dataframes to Excel
        Key = sheet name
        Value = dataframe name '''
    with pd.ExcelWriter(file, engine='xlsxwriter') as writer:
        for sheet, df in df_dict.items():
            rows_to_freeze = df.columns.nlevels
            cols_to_freeze = df.index.nlevels
            df.to_excel(writer,
                        sheet_name=sheet,
                        freeze_panes=(rows_to_freeze, cols_to_freeze),
                        merge_cells=False,
                        **kwargs)

#%% Patent functions
# Reference: https://www.biostars.org/p/357841/

def get_num_patents(cid_list):
    data = {'cid':[],
            'Number of patents':[]}
```

```
    for cid in cid_list:
        data['cid'].append(cid)
        try:
            r = requests.get(f'https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/{cid}/xrefs/PatentID/JSON')
            list_of_patents = r.json()['InformationList']['Information'][0]['PatentID']
            data['Number of patents'].append(len(list_of_patents))
        except:
            data['Number of patents'].append(0)

    df = pd.DataFrame(data)
    return df

def get_patents(cid=None, n_patents=False, print_status=False):
    if (cid==None):
        raise Exception('Must specify parameters')

    if print_status==True:
        print(f'... retrieving patent data for CID {cid}')

    # Get list of patent IDs for a CID
    try:
        r = requests.get(f'https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/{cid}/xrefs/PatentID/JSON')
        list_of_patents = r.json()['InformationList']['Information'][0]['PatentID']

        # Correct formatting on list of patents
        # Add dash after prefix
        list_of_patents_2 = [pat if pat[2] == '-'
                             else pat[:2] + "-" + pat[2:]
                             for pat in list_of_patents]
```

```
        # Add hyphen before last two characters
        list_of_patents_3 = [pat[:-2] + "-" + pat[-2:]
                             # end of patent string does NOT have '-',
                             # AND one of final two characters is non-numeric
                             if (pat[-4:].find('-') == -1)
                                 and (pat[-1].isalpha() or pat[-2].isalpha())
                             else pat
                             for pat in list_of_patents_2]

        # Additional correction
        list_of_patents_4 = [pat[:7] + pat[8:] if pat[:5] == "US-20" and pat[7] == '0'
                             else pat
                             for pat in list_of_patents_3]
        if print_status==True:
            print(f'... ... CID {cid} has {len(list_of_patents_4)} patent records')

        if type(n_patents) == int:
            list_of_patents_4 = list_of_patents_4[:n_patents]

        # create df of patent info for each patent in list
        pat_dfs = {}
        for pat in list_of_patents_4:

            # Empty dataframe for patent
            pat_data = {}
            try:
                pat_json =
requests.get(f'https://pubchem.ncbi.nlm.nih.gov/rest/pug_view/data/patent/{pat}/JSON').json()[
'Record']

                pat_data['Patent number'] = pat_json['RecordAccession']
                pat_data['Title']         = pat_json['RecordTitle']
```

```
                pat_data['URL'] = f'https://pubchem.ncbi.nlm.nih.gov/patent/{pat}'

                # ------------------------------------------------------------
                # 1. find 'Important Dates' section
                sec     = pat_json['Section']
                matches = [x['TOCHeading'] == 'Important Dates' for i, x in enumerate(sec)]
                if True in matches:
                    idx     = matches.index(True)

                    # 2. find 'Priority Date' section
                    sec2    = sec[idx]['Section']
                    matches = [x['TOCHeading'] == 'Priority Date' for i, x in enumerate(sec2)]
                    if True in matches:
                        idx     = matches.index(True)

                        # 3. get priority date
                        pat_data['Priority date'] =
sec2[idx]['Information'][0]['Value']['DateISO8601']

                # ------------------------------------------------------------
                # 2. Abstract
                sec     = pat_json['Section']
                matches = [x['TOCHeading'] == 'Abstract' for i, x in enumerate(sec)]
                if True in matches:
                    idx     = matches.index(True)
                    sec2    = sec[idx]

                    pat_data['Abstract'] =
sec2['Information'][0]['Value']['StringWithMarkup'][0]['String']

                # ------------------------------------------------------------
                # 3. Full text
```

```
                sec     = pat_json['Section']
                matches = [x['TOCHeading'] == 'Full Text' for i, x in enumerate(sec)]
                if True in matches:
                    idx     = matches.index(True)

                    sec2    = sec[idx]['Information']

                    # check for google
                    matches = [x['Name'] == 'Google' for i, x in enumerate(sec2)]
                    # if there is a google link, provide google link
                    if True in matches:
                        idx = matches.index(True)
                        pat_data['Full text'] =
sec2[idx]['Value']['StringWithMarkup'][0]['String']
                    # if no google link, link to PubChem patent page
                    else:
                        pat_data['Full text'] =
f'https://pubchem.ncbi.nlm.nih.gov/patent/{pat}#section=Full-Text'

            # If no patent info retrievable, include only patent number
            except:
                pat_data['Patent number'] = pat

            # Add dataframe to dict of patent dataframes
            pat_dfs[pat] = pd.DataFrame(pat_data, index=[0])

        # Combine all patent dataframes for each CID
        df = pd.concat(pat_dfs.values(), axis=0).assign(cid=cid)

        return df

    # requests to get list of patents for compound fails
```

```
    except:
        print(f'no patents for cid={cid}')


def combine_patent_data(cid_list, n_patents=False, print_status=False):
    ''' Compile single dataframe of additional PubChem information
        for a list of CIDs. Specify URL name and compound name to pass to
        underlying single-chemical function. '''
    if type(cid_list) != list:
        raise Exception('Must specify list format input')

    df = pd.concat([get_patents(c, n_patents=n_patents, print_status=print_status)
                    for c in cid_list],
                    axis=0).reset_index(drop=True)
    return df


#%% ============================================================================
#   IMPORT DATA & RUN FUNCTIONS
#   ============================================================================

#%% Import OFR universe input data

# Starting point: OFR universe file
if pull_ofr_cids == True:
    print('Merging CIDs from PubChem with OFR file')
    OFR_universe = pd.read_csv(ofr_file_raw, low_memory = False)
    OFR_universe.rename({'OFR_CAS':'OFR_CAS_orig'}, axis=1, inplace=True)
    OFR_universe['OFR_CAS'] = OFR_universe['OFR_CAS_orig'].str.replace('t-','')
    # Add CID as column to OFR Universe input file using PubChemPy
    new_col = []
```

```
    tries = 10


    for i, cas in enumerate(OFR_universe['OFR_CAS']):
        print(f'iteration {i} for {cas}')
        for t in range(tries):
            try:
                new_col.append(pcp.get_cids(cas, namespace='name'))
            except:
                if t < tries - 1:
                    print(f'try {t} failed; resting & trying again')
                    time.sleep(15)
                    continue
                else:
                    print(f'max tries reached')
                    raise
            break


    OFR_universe['OFR_cid_as_list'] = new_col
    OFR_universe['OFR_cid'] = OFR_universe['OFR_cid_as_list'].str[0].apply(lambda x:
f"{x:0.0f}")
    OFR_universe.drop('OFR_cid_as_list', axis=1, inplace=True)
    # Move CAS columns to front
    OFR_universe = OFR_universe[[c for c in OFR_universe if c in ['OFR_CAS_orig','OFR_CAS']]
                                + [c for c in OFR_universe if c not in
['OFR_CAS_orig','OFR_CAS']]]


# Alternatively, read from OFR universe with CID from process above already added
else:
    print('Reading OFR file with CIDs already merged')
    OFR_universe = pd.read_excel(ofr_file_with_cids, sheet_name='Sheet1', engine='openpyxl')
    OFR_universe['OFR_CAS'] = OFR_universe['OFR_CAS'].str.replace('t-', '')
    OFR_universe['OFR_cid'] = OFR_universe['OFR_cid'].apply(lambda x: f"{x:0.0f}")
```

```
#%% Run / load PubChem queries to create dictionary of data frames


if query_pubchem == True:


    print('Querying PubChem using custom functions')


    # Chemicals to search

    chems = OFR_universe['OFR_cid'].drop_duplicates().to_list()


    # Start timer ...

    start = time.time()


    # Create empty dataframes in dictionary

    final_dfs = {}

    # Add dataframes

    final_dfs['PubChem'] = combine_pubchem_dfs(chems,
namespace='cid')[['cid','molecular_formula','molecular_weight','xlogp']].copy()


    #final_dfs['PubChem_subset'] =
final_dfs['PubChem'][['cid','molecular_formula','molecular_weight','iupac_name']].copy()

    final_dfs['Synonyms'] = combine_synonyms(chems, namespace='cid')

    final_dfs['Synonyms_long'] = final_dfs['Synonyms'].explode(column='Synonym')

    final_dfs['Refs']    = combine_references(chems)

    # Loop over fields to scrape PubChem for additional data


    for i in additional_data_fields:

        print(f'Pulling {i} data')

        final_dfs[i] = combine_additional_data(chems,

                                               url_name=i,

                                               column_name=i)

    final_dfs['Sections'] = combine_section_check(cid_list=chems,
```

```
                                                section_list=sections_to_query)

    # Either query PubChem for patent info ...
    if query_patents == True:
        # Add dataframe of number of patent records per CID
        final_dfs['Number of patents'] = get_num_patents(chems)

        # Add patent information
        # NOTE: This can be very slow.
        final_dfs['Patents'] = combine_patent_data(chems, print_status=True)

    # Or read in previously-created and saved 'pickle' file
    elif read_patent_pickles == True:
        final_dfs['Number of patents'] = pd.read_pickle('Output/Number of patents_pickle.pkl')
        final_dfs['Patents'] = pd.read_pickle('Output/Patents_pickle.pkl')

    # Ensure each CID column is string for merging
    for k, v in final_dfs.items():
        v['cid'] = v['cid'].astype(str)



    # Additional patent processing
    # Flag patents by keyword,
    # create dataframe of unique CIDs with list of associated patents,
    # and create data frame of unique patents with list of associated CIDs

    # Separate patent dataframe
    pat = final_dfs['Patents'].copy()

    # Match keywords
    flame_ret_keywords = 'flame retard|flame-retard|flameproof|flame-resist|fire retard|fire-
retard|fireproof|fire resist|fire-resist|BFR|OFR'
```

```
    pat['Keyword match (abstract)'] = pat['Abstract'].str.contains(flame_ret_keywords,
case=False, regex=True)

    pat['Keyword match (title & abstract)'] = pat['Abstract'].str.contains(flame_ret_keywords,
case=False, regex=True) & pat['Title'].str.contains(flame_ret_keywords, case=False,
regex=True)


    # Join patents across CIDs

    pat2 = pat.groupby(list(pat.drop('cid', axis='columns').columns),
dropna=False)['cid'].apply(list).reset_index()


    # Create unique CIDs dataframe

    cid = final_dfs['Number of patents'].rename({'Number of patents':'Count of patents in
list'}, axis=1)

    # Summarize patents by CID by summarizing from main patent data

    pat_info_by_cid = pd.concat([

        # patent records found by patent lookup process

        pat.groupby('cid').size().reset_index().rename({0:'Count of patent records returned'},
axis='columns').set_index('cid'),


        # patents with keyword match

        pat.loc[pat['Keyword match
(abstract)'].eq(True)].groupby('cid').size().reset_index().rename({0:'Count of patents with
keywords (abstract)'}, axis='columns').set_index('cid'),

        pat.loc[pat['Keyword match (abstract)'].eq(True)].groupby('cid')['Patent
number'].apply(list).reset_index().rename({'Patent number':'Patents with keyword matches
(abstract)'}, axis='columns').set_index('cid'),


        # patents with keyword match in title *and* abstract

        pat.loc[pat['Keyword match (title &
abstract)'].eq(True)].groupby('cid').size().reset_index().rename({0:'Count of patents with
keywords (title & abstract)'}, axis='columns').set_index('cid'),

        pat.loc[pat['Keyword match (title & abstract)'].eq(True)].groupby('cid')['Patent
number'].apply(list).reset_index().rename({'Patent number':'Patents with keyword matches
(title & abstract)'}, axis='columns').set_index('cid')

        ], axis=1)
```

```
    cid2 = cid.merge(pat_info_by_cid, on='cid', how='left')


    # Add to final DFs
    final_dfs['Patent info by patent'] = pat2.drop('URL', axis=1).loc[pat2['Keyword match
(abstract)'].eq(True)].reset_index(drop=True)
    final_dfs['Patent info by CID'] = cid2


    # ============================================================================
    # Save all dataframes to python 'pickle' files
    for k, v in final_dfs.items():
        v.to_pickle(f'Output/{k}_pickle.pkl')
    # ============================================================================


    # ... end timer
    end = time.time()
    print(f'took {(end-start) / 60} minutes')



# Load data from pickle files
if query_pubchem == False:
    print('Using saved PubChem pickle files')


    # Read pickle into dict
    pubchem_data = {}
    for file in os.listdir('Output/'):
        if file.endswith(".pkl"):
            pubchem_data[file.split('_pickle')[0]] = pd.read_pickle(f'Output/{file}')
    final_dfs = pubchem_data


#%% ============================================================================
#   EXPORT RESULTS
#   ============================================================================
```

```
#%% Final data processing
# For size considerations, drop all patent records from exported dataframes
patents = final_dfs.pop('Patents')
# Basic patent counts are reflected in 'Patent info by CID' dataframe
patent_stats = final_dfs.pop('Number of patents')

# convert CIDs into object
for k in final_dfs.keys():
    final_dfs[k]['cid'] = final_dfs[k]['cid'].astype(str)

OFR_universe['OFR_CAS'] = 't-' + OFR_universe['OFR_CAS']

# Merge each dataframe with OFR input file in new dictionary
final_dfs_ofr = {key:OFR_universe.merge(final_dfs[key],
                                        how='left',
                                        left_on='OFR_cid',
                                        right_on='cid')
                 for key in final_dfs.keys()
                 if key not in ['Patent info by patent']}

# Add patent info by patent *not* merged to OFR CIDs
final_dfs_ofr['Patent info by patent'] = final_dfs['Patent info by patent']

#%% Export to Excel
# Reorder dictionary
key_order = ['PubChem',
             'Synonyms',
             'Synonyms_long',
             'Boiling Point',
             'Flash-Point',
```

```
            'Henrys-Law-Constant',
            'LogKoa',
            'LogP',
            'Solubility',
            'Vapor Pressure',
            'Sections',
            'Refs',
            'Patent info by CID',
            'Patent info by patent']
final_dfs_ofr_ordered = {k:final_dfs_ofr[k] for k in key_order}
# Assert that no items have been dropped
assert len(final_dfs_ofr) == len(final_dfs_ofr_ordered)

# List of OFR chemicals & CIDs if generating new list
if pull_ofr_cids == True:
    (OFR_universe.drop('OFR_CAS', axis=1)
                 .rename({'OFR_CAS_orig':'OFR_CAS'}, axis=1)
                 .to_excel(f'Output/OFR_universe_cid_{time.strftime("%m%d%Y")}.xlsx',
                           freeze_panes=(1,1),
                           index=False))

export_dfs(f'Output/OFR_universe_wPCPy_{time.strftime("%m%d%Y")}.xlsx',
           final_dfs_ofr_ordered,
           index=False)
```

A.2 PYTHON CODE FOR PRODUCING DATA SOURCE OUTPUTS

```
# -*- coding: utf-8 -*-
"""
Constructing OFR Database and supporting output files from multiple data sources.

Created on Mon Jun 21 18:44:36 2021
```

```
@author: RCabral
"""

#%% Import modules.
# Os allows file path operations.
import os
# Pandas contains "data frame" structure and data manipulation processes.
import pandas as pd
# Allows use of date in file exports.
from datetime import date
# Numpy contains many numerical operations
import numpy as np

#%% Define file paths and save status.
# Root paths:
BASE_INPUT_PATH = os.path.join(os.path.dirname(os.path.realpath('__file__')), 'Input')
OUTPUT_PATH = os.path.join(os.path.dirname(os.path.realpath('__file__')), 'Output')

# !!!
# Do you want to save files this run?
# True = Save files this run.
# False = Do not save files this run.

save_toggle = 'False'

#%% Import data sources.
OFR_universe = pd.read_csv(os.path.join(BASE_INPUT_PATH, r'OFR_universe_cid_03042022.csv'),
low_memory = False)

echa_add = pd.read_csv(os.path.join(BASE_INPUT_PATH,r'8_EuropeanData\Pages from Mapping
exercise – Plastic additives initiative - ECHA_FRs.csv'), low_memory = False)
```

```
echa_links = pd.read_csv(os.path.join(BASE_INPUT_PATH,r'ECHA_links\ECHA Advanced Search Br and Cl_Consumer Use Article Service Life_09272021.csv'), low_memory = False, encoding='cp1252')


comptox_batch = pd.read_csv(os.path.join(BASE_INPUT_PATH, r'CompTox\CCD-Batch-Search_2021-12-08_02_23_41.csv'), low_memory = False)


comptox_syns = pd.read_csv(os.path.join(BASE_INPUT_PATH, r'CompTox\itdatahub_36_synonyms_with_quality_list_only.csv'), low_memory = False)


comptox_exp = pd.read_csv(os.path.join(BASE_INPUT_PATH, r'CompTox\itdatahub_37_chemical_property_list_only.csv'), low_memory = False)


comptox_pred = pd.read_csv(os.path.join(BASE_INPUT_PATH, r'CompTox\NCD_Dashboard_TEST and OPERA__2021-06-18_22_15_23.csv'), low_memory = False)


norman = pd.read_csv(os.path.join(BASE_INPUT_PATH, r'Norman\susdat_2021-06-04-180832.csv'), low_memory = False)


pc_py = pd.read_excel(os.path.join(BASE_INPUT_PATH, r'PubChem_py\OFR_universe_wPCPy_03042022.xlsx'), sheet_name = None)

del pc_py['Patent info by patent'] #Remove sheet that's not relevant for pairing OFR information.


pc_brows_id = pd.read_csv(os.path.join(BASE_INPUT_PATH, r'PubChem_browser\PubChem_identification.csv'),low_memory = False)


pc_brows_mona = pd.read_csv(os.path.join(BASE_INPUT_PATH, r'PubChem_browser\PubChem_infosources_MoNA.csv'), low_memory = False)


pc_brows_norman = pd.read_csv(os.path.join(BASE_INPUT_PATH, r'PubChem_browser\PubChem_ontology_NORMAN.csv'), low_memory = False)


pc_brows_ms = pd.read_csv(os.path.join(BASE_INPUT_PATH, r'PubChem_browser\PubChem_spectralinfo_MS.csv'), low_memory = False)
```

```
qsur = pd.read_csv(os.path.join(BASE_INPUT_PATH,
r'QSUR\fr_ofr_qsur_predictions_03042022.csv'), low_memory = False)


tri = pd.read_csv(os.path.join(BASE_INPUT_PATH, r'TRI\ry_2020_tri_chemical_synonyms.csv'),
encoding='cp1252', low_memory = False)


tsca = pd.read_csv(os.path.join(BASE_INPUT_PATH, r'TSCA\TSCAINV_022021.csv'), low_memory =
False)


#%% CPDat files need an extra layer of pre-processing.

# Import CPDat tables and create dataframes.

# Specify directory where csv files are.

csvs = [x for x in os.listdir(os.path.join(BASE_INPUT_PATH,r'CompTox\CPDATRelease20201216'))
if x.endswith('.csv')]


# Grab file name to use as dataframe name.

fns = [os.path.splitext(os.path.basename(x))[0] for x in csvs]


d = {}


for i in range(len(fns)):

    d[fns[i]] = pd.read_csv(os.path.join(BASE_INPUT_PATH,r'CompTox\CPDatRelease20201216',
csvs[i]), encoding='cp1252', index_col = False, low_memory=False)



# Get fields of interest into one table.

ingredients = d['product_composition_data_20201216']

products = d['PUC_dictionary_20201216']

substances = d['chemical_dictionary_20201216']


comptox_prod = substances.merge(ingredients, how='left',
on='chemical_id').merge(products,how='left',on='puc_id')


#%% Define varibles and formatting.
```

```
today = date.today()

d1 = today.strftime('%m%d%Y')


# Prior to import, CSV file decides to read some CAS numbers as dates.

OFR_universe['OFR_CAS'] = OFR_universe['OFR_CAS'].astype('string') # Make sure column is being seen as string

OFR_universe['OFR_CAS'] = OFR_universe['OFR_CAS'].str.replace('t-','') # Fix formatting

OFR_universe['OFR_DTXSID'] = OFR_universe['OFR_DTXSID'].astype('string') # Make sure column is being seen as string

OFR_universe.replace({'-':np.nan}, inplace=True)


echa_add['CAS_number'] = echa_add['CAS_number'].astype('string') # Make sure column is being seen as string

echa_add.replace({'-':np.nan}, inplace=True)


echa_links['CASNumber'] = echa_links['CASNumber'].astype('string') # Make sure column is being seen as string

echa_links['CASNumber'] = echa_links['CASNumber'].str.replace('t-','') # Fix formatting

echa_links.replace({'-':np.nan}, inplace=True)


comptox_batch['DTXSID'] = comptox_batch['DTXSID'].astype('string') # Make sure column is being seen as string

comptox_batch.replace({'-':np.nan}, inplace=True)

comptox_batch.replace({'-':np.nan}, inplace=True)


comptox_syns['dtxsid'] = comptox_syns['dtxsid'].astype('string') # Make sure column is being seen as string

comptox_syns['dtxsid'] = comptox_syns['dtxsid'].str.replace(' ','') # Fix formatting.

comptox_syns.replace({'-':np.nan}, inplace=True)


comptox_exp['dtxsid'] = comptox_exp['dtxsid'].astype('string') # Make sure column is being seen as string

comptox_exp['dtxsid'] = comptox_exp['dtxsid'].str.replace(' ','') # Fix formatting.

comptox_exp.replace({'-':np.nan}, inplace=True)
```

```
comptox_pred['DTXSID'] = comptox_pred['DTXSID'].astype('string') # Make sure column is being seen as string
comptox_pred.replace({'-':np.nan}, inplace=True)

comptox_prod['DTXSID'] = comptox_prod['DTXSID'].astype('string') # Make sure column is being seen as string
comptox_pred.replace({'-':np.nan}, inplace=True)

norman['CAS_RN'] = norman['CAS_RN'].astype('string') # Make sure column is being seen as string
norman['CAS_RN'] = norman['CAS_RN'].str.replace('CAS_RN: ','') # Fix formatting
norman.replace({'-':np.nan}, inplace=True)

pc_brows_id['cid'] = pc_brows_id['cid'].astype('int32') # Make sure column is being seen as numeric
pc_brows_id.replace({'-':np.nan}, inplace=True)

pc_brows_mona['cid'] = pc_brows_mona['cid'].astype('int32') # Make sure column is being seen as numeric
pc_brows_mona.replace({'-':np.nan}, inplace=True)

pc_brows_norman['cid'] = pc_brows_norman['cid'].astype('int32') # Make sure column is being seen as numeric
pc_brows_norman.replace({'-':np.nan}, inplace=True)

pc_brows_ms['cid'] = pc_brows_ms['cid'].astype('int32') # Make sure column is being seen as numeric
pc_brows_ms.replace({'-':np.nan}, inplace=True)

qsur['dtxsid'] = qsur['dtxsid'].astype('string') # Make sure column is being seen as string
qsur.replace({'-':np.nan}, inplace=True)

tri['CAS'] = tri['CASRN'].astype('string') # Make sure column is being seen as string
```

```
tri['CAS'] = tri['CASRN'].str.replace('t-','') # Fix formatting.

tri.replace({'-':np.nan}, inplace=True)


tsca['CASRN'] = tsca['CASRN'].astype('string') # Make sure column is being seen as string

tsca.replace({'-':np.nan}, inplace=True)


#%% Filter each data source to our OFRs and pair with OFR_Universe.

OFR_universe_wecha_add = OFR_universe.merge(echa_add[pd.notnull(echa_add.CAS_number)],
how='left', left_on='OFR_CAS', right_on='CAS_number')

echa_add_fields = list(echa_add) # Get existing column names

new_fields = ['ECHAadd_' + s for s in echa_add_fields] # Define new column names

d_new_fields = {echa_add_fields[i]: new_fields[i] for i in range(len(echa_add_fields))} #
Create dictionary for updating them

OFR_universe_wecha_add.rename(columns = d_new_fields, inplace=True) # Update field names


OFR_universe_wecha_links = OFR_universe.merge(echa_links[pd.notnull(echa_links.CASNumber)],
how='left', left_on='OFR_CAS', right_on='CASNumber')

echa_links_fields = list(echa_links) # Get existing column names

new_fields = ['ECHAlinks_' + s for s in echa_links_fields] # Define new column names

d_new_fields = {echa_links_fields[i]: new_fields[i] for i in range(len(echa_links_fields))} #
Create dictionary for updating them

OFR_universe_wecha_links.rename(columns = d_new_fields, inplace=True) # Update field names


OFR_universe_wcomptox_batch =
OFR_universe.merge(comptox_batch[pd.notnull(comptox_batch.DTXSID)], how='left',
left_on='OFR_DTXSID', right_on='DTXSID')

comptox_batch_fields = list(comptox_batch) # Get existing column names

new_fields = ['CompTox_' + s for s in comptox_batch_fields] # Define new column names

d_new_fields = {comptox_batch_fields[i]: new_fields[i] for i in
range(len(comptox_batch_fields))} # Create dictionary for updating them

OFR_universe_wcomptox_batch.rename(columns = d_new_fields, inplace=True) # Update field names


OFR_universe_wcomptox_syns = OFR_universe.merge(comptox_syns[pd.notnull(comptox_syns.dtxsid)],
how='left', left_on='OFR_DTXSID', right_on='dtxsid')

comptox_syns_fields = list(comptox_syns) # Get existing column names
```

```
new_fields = ['CompTox_' + s for s in comptox_syns_fields] # Define new column names

d_new_fields = {comptox_syns_fields[i]: new_fields[i] for i in
range(len(comptox_syns_fields))} # Create dictionary for updating them

OFR_universe_wcomptox_syns.rename(columns = d_new_fields, inplace=True) # Update field names


OFR_universe_wcomptox_exp = OFR_universe.merge(comptox_exp[pd.notnull(comptox_exp.dtxsid)],
how='left', left_on='OFR_DTXSID', right_on='dtxsid')

comptox_exp_fields = list(comptox_exp) # Get existing column names

new_fields = ['CompTox_' + s for s in comptox_exp_fields] # Define new column names

d_new_fields = {comptox_exp_fields[i]: new_fields[i] for i in range(len(comptox_exp_fields))}
# Create dictionary for updating them

OFR_universe_wcomptox_exp.rename(columns = d_new_fields, inplace=True) # Update field names


OFR_universe_wcomptox_pred = OFR_universe.merge(comptox_pred[pd.notnull(comptox_pred.DTXSID)],
how='left', left_on='OFR_DTXSID', right_on='DTXSID')

comptox_pred_fields = list(comptox_pred) # Get existing column names

new_fields = ['CompTox_' + s for s in comptox_pred_fields] # Define new column names

d_new_fields = {comptox_pred_fields[i]: new_fields[i] for i in
range(len(comptox_pred_fields))} # Create dictionary for updating them

OFR_universe_wcomptox_pred.rename(columns = d_new_fields, inplace=True) # Update field names


OFR_universe_wcomptox_prod = OFR_universe.merge(comptox_prod[pd.notnull(comptox_prod.DTXSID)],
how='left', left_on='OFR_DTXSID', right_on='DTXSID')

comptox_prod_fields = list(comptox_prod) # Get existing column names

new_fields = ['CompTox_' + s for s in comptox_prod_fields] # Define new column names

d_new_fields = {comptox_prod_fields[i]: new_fields[i] for i in
range(len(comptox_prod_fields))} # Create dictionary for updating them

OFR_universe_wcomptox_prod.rename(columns = d_new_fields, inplace=True) # Update field names


OFR_universe_wnorman = OFR_universe.merge(norman[pd.notnull(norman.CAS_RN)], how='left',
left_on='OFR_CAS', right_on='CAS_RN')

norman_fields = list(norman) # Get existing column names

new_fields = ['Norman_' + s for s in norman_fields] # Define new column names

d_new_fields = {norman_fields[i]: new_fields[i] for i in range(len(norman_fields))} # Create
dictionary for updating them
```

```
OFR_universe_wnorman.rename(columns = d_new_fields, inplace=True) # Update field names


OFR_universe_wpc_brows_id = OFR_universe.merge(pc_brows_id[pd.notnull(pc_brows_id.cid)],
how='left', left_on='OFR_cid', right_on='cid')

pc_brows_id_fields = list(pc_brows_id) # Get existing column names

new_fields = ['PCBrowser_' + s for s in pc_brows_id_fields] # Define new column names

d_new_fields = {pc_brows_id_fields[i]: new_fields[i] for i in range(len(pc_brows_id_fields))}
# Create dictionary for updating them

OFR_universe_wpc_brows_id.rename(columns = d_new_fields, inplace=True) # Update field names


OFR_universe_wpc_brows_mona = OFR_universe.merge(pc_brows_mona[pd.notnull(pc_brows_mona.cid)],
how='left', left_on='OFR_cid', right_on='cid')

pc_brows_mona_fields = list(pc_brows_mona) # Get existing column names

new_fields = ['PCBrowser_' + s for s in pc_brows_mona_fields] # Define new column names

d_new_fields = {pc_brows_mona_fields[i]: new_fields[i] for i in
range(len(pc_brows_mona_fields))} # Create dictionary for updating them

OFR_universe_wpc_brows_mona.rename(columns = d_new_fields, inplace=True) # Update field names


OFR_universe_wpc_brows_norman =
OFR_universe.merge(pc_brows_norman[pd.notnull(pc_brows_norman.cid)], how='left',
left_on='OFR_cid', right_on='cid')

pc_brows_norman_fields = list(pc_brows_norman) # Get existing column names

new_fields = ['PCBrowser_' + s for s in pc_brows_norman_fields] # Define new column names

d_new_fields = {pc_brows_norman_fields[i]: new_fields[i] for i in
range(len(pc_brows_norman_fields))} # Create dictionary for updating them

OFR_universe_wpc_brows_norman.rename(columns = d_new_fields, inplace=True) # Update field
names


OFR_universe_wpc_brows_ms = OFR_universe.merge(pc_brows_ms[pd.notnull(pc_brows_ms.cid)],
how='left', left_on='OFR_cid', right_on='cid')

pc_brows_ms_fields = list(pc_brows_ms) # Get existing column names

new_fields = ['PCBrowser_' + s for s in pc_brows_ms_fields] # Define new column names

d_new_fields = {pc_brows_ms_fields[i]: new_fields[i] for i in range(len(pc_brows_ms_fields))}
# Create dictionary for updating them

OFR_universe_wpc_brows_ms.rename(columns = d_new_fields, inplace=True) # Update field names
```

```
OFR_universe_wqsur = OFR_universe.merge(qsur[pd.notnull(qsur.dtxsid)], how='left',
left_on='OFR_DTXSID', right_on='dtxsid')

qsur_fields = list(qsur) # Get existing column names

new_fields = ['QSUR_' + s for s in qsur_fields] # Define new column names

d_new_fields = {qsur_fields[i]: new_fields[i] for i in range(len(qsur_fields))} # Create
dictionary for updating them

OFR_universe_wqsur.rename(columns = d_new_fields, inplace=True) # Update field names


OFR_universe_wTRI = OFR_universe.merge(tri[pd.notnull(tri.CAS)], how='left',
left_on='OFR_CAS', right_on='CAS')

tri_fields = list(tri) # Get existing column names

new_fields = ['TRI_' + s for s in tri_fields] # Define new column names

d_new_fields = {tri_fields[i]: new_fields[i] for i in range(len(tri_fields))} # Create
dictionary for updating them

OFR_universe_wTRI.rename(columns = d_new_fields, inplace=True) # Update field names


OFR_universe_wTSCA = OFR_universe.merge(tsca[pd.notnull(tsca.CASRN)], how='left',
left_on='OFR_CAS', right_on='CASRN')

tsca_fields = list(tsca) # Get existing column names

new_fields = ['TSCA_' + s for s in tsca_fields] # Define new column names

d_new_fields = {tsca_fields[i]: new_fields[i] for i in range(len(tsca_fields))} # Create
dictionary for updating them

OFR_universe_wTSCA.rename(columns = d_new_fields, inplace=True) # Update field names


#%% Transpose results to Y/N list by data source

set_echa_add = set(OFR_universe_wecha_add['ECHAadd_CAS_number'].drop_duplicates().dropna())

set_echa_links =
set(OFR_universe_wecha_links['ECHAlinks_CASNumber'].drop_duplicates().dropna())

set_comptox_batch =
set(OFR_universe_wcomptox_batch['CompTox_DTXSID'].drop_duplicates().dropna())

set_comptox_syns =
set(OFR_universe_wcomptox_syns['CompTox_dtxsid'].drop_duplicates().dropna())

set_comptox_exp = set(OFR_universe_wcomptox_exp['CompTox_dtxsid'].drop_duplicates().dropna())
```

```
set_comptox_pred =
set(OFR_universe_wcomptox_pred['CompTox_DTXSID'].drop_duplicates().dropna())

set_comptox_prod =
set(OFR_universe_wcomptox_prod['CompTox_DTXSID'].drop_duplicates().dropna())

set_norman = set(OFR_universe_wnorman['Norman_CAS_RN'].drop_duplicates().dropna())

set_pc_brows_id = set(OFR_universe_wpc_brows_id['PCBrowser_cid'].drop_duplicates().dropna())

set_pc_brows_mona =
set(OFR_universe_wpc_brows_mona['PCBrowser_cid'].drop_duplicates().dropna())

set_pc_brows_norman =
set(OFR_universe_wpc_brows_norman['PCBrowser_cid'].drop_duplicates().dropna())

set_pc_brows_ms = set(OFR_universe_wpc_brows_ms['PCBrowser_cid'].drop_duplicates().dropna())

set_qsur = set(OFR_universe_wqsur['QSUR_dtxsid'].drop_duplicates().dropna())

set_TRI = set(OFR_universe_wTRI['TRI_CAS'].drop_duplicates().dropna())

set_tsca = set(OFR_universe_wTSCA['TSCA_CASRN'].drop_duplicates().dropna())


# Compare to the OFR list and export the results.

SourcesList = OFR_universe.copy()


SourcesList['ECHA_add'] = SourcesList['OFR_CAS'].apply(lambda x: 'Y' if x in set_echa_add else
'-')

SourcesList['ECHA_links'] = SourcesList['OFR_CAS'].apply(lambda x: 'Y' if x in set_echa_links
else '-')

SourcesList['CompTox_batch'] = SourcesList['OFR_DTXSID'].apply(lambda x: 'Y' if x in
set_comptox_batch else '-')

SourcesList['CompTox_syns'] = SourcesList['OFR_DTXSID'].apply(lambda x: 'Y' if x in
set_comptox_syns else '-')

SourcesList['CompTox_exp'] = SourcesList['OFR_DTXSID'].apply(lambda x: 'Y' if x in
set_comptox_exp else '-')

SourcesList['CompTox_pred'] = SourcesList['OFR_DTXSID'].apply(lambda x: 'Y' if x in
set_comptox_pred else '-')

SourcesList['CompTox_prod'] = SourcesList['OFR_DTXSID'].apply(lambda x: 'Y' if x in
set_comptox_prod else '-')

SourcesList['Norman'] = SourcesList['OFR_CAS'].apply(lambda x: 'Y' if x in set_norman else '-
')

SourcesList['PC_Browser_id'] = SourcesList['OFR_cid'].apply(lambda x: 'Y' if x in
set_pc_brows_id else '-')
```

```
SourcesList['PC_Browser_mona'] = SourcesList['OFR_cid'].apply(lambda x: 'Y' if x in
set_pc_brows_mona else '-')

SourcesList['PC_Browser_norman'] = SourcesList['OFR_cid'].apply(lambda x: 'Y' if x in
set_pc_brows_norman else '-')

SourcesList['PC_Browser_ms'] = SourcesList['OFR_cid'].apply(lambda x: 'Y' if x in
set_pc_brows_ms else '-')

SourcesList['QSUR'] = SourcesList['OFR_DTXSID'].apply(lambda x: 'Y' if x in set_qsur else '-')

SourcesList['TRI'] = SourcesList['OFR_CAS'].apply(lambda x: 'Y' if x in set_TRI else '-')

SourcesList['TSCA'] = SourcesList['OFR_CAS'].apply(lambda x: 'Y' if x in set_tsca else '-')


# Add PubChemPy Output to the data frame of sources

pc_py_sheets = list(pc_py.keys())


for i in range(len(pc_py_sheets)):

    df_name = pc_py_sheets[i]

    df_temp = pc_py[df_name]

    set_temp = set(df_temp['cid'].drop_duplicates().dropna().astype(str))

    SourcesList[f'PCPy_{df_name}'] = SourcesList['OFR_cid'].astype(str).apply(lambda x: 'Y' if
x in set_temp else '-')


# !!!

if save_toggle == 'True':

    SourcesList.to_csv(os.path.join(OUTPUT_PATH, f'support\SourcesList_{d1}.csv'), index =
False)

    print('Sources list saved.')


elif save_toggle == 'False':

    print('No sources list saved this time.')


else:

    print('Invalid save_toggle input.')



#%% Add "t-" back to CAS numbers before export.
```

```
# Prior to import, CSV file decides to read some CAS numbers as dates.


OFR_universe['OFR_CAS'] = "t-" + OFR_universe['OFR_CAS']


OFR_universe_wecha_add['OFR_CAS'] = "t-" + OFR_universe_wecha_add['OFR_CAS']
OFR_universe_wecha_links['OFR_CAS'] = "t-" + OFR_universe_wecha_links['OFR_CAS']
OFR_universe_wcomptox_batch['OFR_CAS'] = "t-" + OFR_universe_wcomptox_batch['OFR_CAS']
OFR_universe_wcomptox_syns['OFR_CAS'] = "t-" + OFR_universe_wcomptox_syns['OFR_CAS']
OFR_universe_wcomptox_exp['OFR_CAS'] = "t-" + OFR_universe_wcomptox_exp['OFR_CAS']
OFR_universe_wcomptox_pred['OFR_CAS'] = "t-" + OFR_universe_wcomptox_pred['OFR_CAS']
OFR_universe_wcomptox_prod['OFR_CAS'] = "t-" + OFR_universe_wcomptox_prod['OFR_CAS']
OFR_universe_wnorman['OFR_CAS'] = "t-" + OFR_universe_wnorman['OFR_CAS']
OFR_universe_wpc_brows_id['OFR_CAS'] = "t-" + OFR_universe_wpc_brows_id['OFR_CAS']
OFR_universe_wpc_brows_mona['OFR_CAS'] = "t-" + OFR_universe_wpc_brows_mona['OFR_CAS']
OFR_universe_wpc_brows_norman['OFR_CAS'] = "t-" + OFR_universe_wpc_brows_norman['OFR_CAS']
OFR_universe_wpc_brows_ms['OFR_CAS'] = "t-" + OFR_universe_wpc_brows_ms['OFR_CAS']
OFR_universe_wqsur['OFR_CAS'] = "t-" + OFR_universe_wqsur['OFR_CAS']
OFR_universe_wTRI['OFR_CAS'] = "t-" + OFR_universe_wTRI['OFR_CAS']
OFR_universe_wTSCA['OFR_CAS'] = "t-" + OFR_universe_wTSCA['OFR_CAS']


#%% Export resulting tables.
#!!!
if save_toggle == 'True':
    OFR_universe_wecha_add.to_csv(os.path.join(OUTPUT_PATH,
f'OFR_universe_wECHA_add_{d1}.csv'), index = False)
    OFR_universe_wecha_links.to_csv(os.path.join(OUTPUT_PATH,
f'OFR_universe_wECHA_links_{d1}.csv'), index = False)
    OFR_universe_wcomptox_batch.to_csv(os.path.join(OUTPUT_PATH,
f'OFR_universe_wCompTox_batch_{d1}.csv'), index = False)
    OFR_universe_wcomptox_syns.to_csv(os.path.join(OUTPUT_PATH,
f'OFR_universe_wCompTox_syns_{d1}.csv'), index = False)
    OFR_universe_wcomptox_exp.to_csv(os.path.join(OUTPUT_PATH,
f'OFR_universe_wCompTox_exp_{d1}.csv'), index = False)
```

```
    OFR_universe_wcomptox_pred.to_csv(os.path.join(OUTPUT_PATH,
f'OFR_universe_wCompTox_pred_{d1}.csv'), index = False)

    OFR_universe_wcomptox_prod.to_csv(os.path.join(OUTPUT_PATH,
f'OFR_universe_wCompTox_prod_{d1}.csv'), index = False)

    OFR_universe_wnorman.to_csv(os.path.join(OUTPUT_PATH, f'OFR_universe_wNorman_{d1}.csv'),
index = False)

    OFR_universe_wpc_brows_id.to_csv(os.path.join(OUTPUT_PATH,
f'OFR_universe_wPCBrowser_ID_{d1}.csv'), index = False)

    OFR_universe_wpc_brows_mona.to_csv(os.path.join(OUTPUT_PATH,
f'OFR_universe_wPCBrowser_MoNA_{d1}.csv'), index = False)

    OFR_universe_wpc_brows_norman.to_csv(os.path.join(OUTPUT_PATH,
f'OFR_universe_wPCBrowser_Norman_{d1}.csv'), index = False)

    OFR_universe_wpc_brows_ms.to_csv(os.path.join(OUTPUT_PATH,
f'OFR_universe_wPCBrowser_MS_{d1}.csv'), index = False)

    OFR_universe_wqsur.to_csv(os.path.join(OUTPUT_PATH, f'OFR_universe_wQSUR_{d1}.csv'), index
= False)

    OFR_universe_wTRI.to_csv(os.path.join(OUTPUT_PATH, f'OFR_universe_wTRI_{d1}.csv'), index =
False)

    OFR_universe_wTSCA.to_csv(os.path.join(OUTPUT_PATH, f'OFR_universe_wTSCA_{d1}.csv'), index
= False)

    print('Data output files saved.')


elif save_toggle == 'False':

    print('No data output files saved this time.')


else:

    print('Invalid save_toggle input.')


#%% Create dataframe of field names and export it

# Pull file names from directory where saved exports are

fnslist = [x for x in os.listdir(os.path.join(OUTPUT_PATH)) if x.endswith('.csv')]

df_fns = pd.DataFrame()


for i in range(len(fnslist)):

    # read the csv file using read_csv
```

```
    # store the data frame in variable called df
    df = pd.read_csv(os.path.join(OUTPUT_PATH,fnslist[i]), index_col=False, low_memory=False)

    # create a list of column names by calling the .columns
    list_of_column_names = list(df.columns)
    df_of_column_names = pd.DataFrame(list_of_column_names, columns=[fnslist[i]])

    # add to dataframe
    df_fns = pd.concat([df_fns,df_of_column_names],axis=1)

# Add PubChemPy Output to the data frame of field names
pc_py_sheets = list(pc_py.keys())
df_pcpy = pd.DataFrame()

for i in range(len(pc_py_sheets)):
    # extract the field names from the dictionary
    list_of_column_names = list(pc_py.get(pc_py_sheets[i]))
    df_of_column_names = pd.DataFrame(list_of_column_names, columns=[pc_py_sheets[i]])

    # add them to the data frame for PubChemPy
    df_pcpy = pd.concat([df_pcpy,df_of_column_names], axis=1)

new_fields = ['PCPy_' + s for s in pc_py_sheets] # Define new column names
d_new_fields = {pc_py_sheets[i]: new_fields[i] for i in range(len(pc_py_sheets))} # Create
dictionary for updating them
df_pcpy.rename(columns = d_new_fields, inplace=True) # Update field names

# Merge the PubChemPy data frame to the main data frame of field names.
frames = [df_fns, df_pcpy]
df_fns_wpcpy = pd.concat(frames, axis = 1)

# !!!
if save_toggle == 'True':
```

```
    df_fns_wpcpy.to_csv(os.path.join(OUTPUT_PATH, f'support\FieldsList_{d1}.csv'), index =
False)
    print('Fields list saved.')


elif save_toggle == 'False':
    print('No fields list saved this time.')


else:
    print('Invalid save_toggle input.')


#%% Create export of field counts
# !!!
if save_toggle == 'True':
    df_fns_wpcpy.to_csv(os.path.join(OUTPUT_PATH, f'support\FieldsList_{d1}.csv'), index =
False)
    # Create dictionary of primary IDs used for each data source.
    prime_ids = {f'OFR_universe_wCompTox_batch_{d1}.csv':'OFR_DTXSID',
      f'OFR_universe_wCompTox_exp_{d1}.csv':'OFR_DTXSID',
      f'OFR_universe_wCompTox_pred_{d1}.csv':'OFR_DTXSID',
      f'OFR_universe_wCompTox_prod_{d1}.csv':'OFR_DTXSID',
      f'OFR_universe_wCompTox_syns_{d1}.csv':'OFR_DTXSID',
      f'OFR_universe_wECHA_add_{d1}.csv':'OFR_CAS',
      f'OFR_universe_wECHA_links_{d1}.csv':'OFR_CAS',
      f'OFR_universe_wNorman_{d1}.csv':'OFR_CAS',
      f'OFR_universe_wPCBrowser_ID_{d1}.csv':'OFR_cid',
      f'OFR_universe_wPCBrowser_MoNA_{d1}.csv':'OFR_cid',
      f'OFR_universe_wPCBrowser_MS_{d1}.csv':'OFR_cid',
      f'OFR_universe_wPCBrowser_Norman_{d1}.csv':'OFR_cid',
      f'OFR_universe_wQSUR_{d1}.csv':'OFR_DTXSID',
      f'OFR_universe_wTRI_{d1}.csv':'OFR_CAS',
      f'OFR_universe_wTSCA_{d1}.csv':'OFR_CAS'}


    # Pull file names from directory where saved exports are
```

```
    fnslist = [x for x in os.listdir(os.path.join(OUTPUT_PATH)) if x.endswith('.csv')]
    df_fns = pd.DataFrame()

    # Compare counts of records associated with each valid value in each column.
    # The <with> loop allows the writing operation to occur over the entire loop
    # then closes the writer after the completion of the <for> loop
    with pd.ExcelWriter(os.path.join(OUTPUT_PATH, f'support\FieldCounts_{d1}.xlsx'),
engine='xlsxwriter') as writer:

        for i in range(len(fnslist)):
            # read the csv file using read_csv
            # store the data frame in variable called temp_df
            temp_fn = fnslist[i]
            df_temp = pd.read_csv(os.path.join(OUTPUT_PATH,temp_fn), index_col=False,
low_memory=False)
            df_temp_count = df_temp.count().to_frame().rename(columns={0:'Count_Non_NaN'})

            cols = list(df_temp.columns)
            df_id_count = pd.DataFrame([])

            for i, c in enumerate(cols):
                id_count =
df_temp[prime_ids[temp_fn]].loc[df_temp[c].notna()].to_frame().drop_duplicates().count()
                temp_dict_id_count = {'Field_Names':cols[i], 'Primary_ID':
id_count.index.tolist(), 'Count_Unique_Primary_ID': id_count[0]}
                df_temp_id_count =
pd.DataFrame(data=temp_dict_id_count).set_index('Field_Names')
                df_id_count = df_id_count.append(df_temp_id_count)

            df_temp = df_temp.astype(str).apply(lambda x: x.str.lower()).replace('nan',np.nan)
            df_temp_unique =
df_temp.nunique(dropna=True).to_frame().rename(columns={0:'Count_Unique_Vals'})
```

```
            df_temp_concat = pd.concat([df_temp_count,df_temp_unique,df_id_count], axis =1,
join='inner')
            df_temp_concat.to_excel(writer, sheet_name=f'{temp_fn[:30]}')


        # Add PubChemPy Output to the FieldCounts support file.
        pc_py_sheets = list(pc_py.keys())


        for i in range(len(pc_py_sheets)):
            df_name = pc_py_sheets[i]
            df_temp = pc_py[df_name]
            df_temp_count = df_temp.count().to_frame().rename(columns={0:'Count_Non_NaN'})


            cols = list(df_temp.columns)
            df_id_count = pd.DataFrame([])


            for i, c in enumerate (cols):
                id_count =
df_temp['OFR_cid'].loc[df_temp[c].notna()].to_frame().drop_duplicates().count()
                temp_dict_id_count = {'Field_Names':cols[i], 'Primary_ID':
id_count.index.tolist(), 'Count_Unique_Primary_ID': id_count[0]}
                df_temp_id_count =
pd.DataFrame(data=temp_dict_id_count).set_index('Field_Names')
                df_id_count = df_id_count.append(df_temp_id_count)


            df_temp = df_temp.astype(str).apply(lambda x: x.str.lower()).replace('nan',np.nan)
            df_temp_unique =
df_temp.nunique(dropna=True).to_frame().rename(columns={0:'Count_Unique_Vals'})


            df_temp_concat = pd.concat([df_temp_count,df_temp_unique,df_id_count], axis =1,
join='inner')
            df_temp_concat.to_excel(writer, sheet_name=f'PCPy_{df_name}')
    print('Field counts saved.')


elif save_toggle == 'False':
```

```
    print('No field counts saved this time.')


else:
    print('Invalid save_toggle input.')
```