D:\NHRI\AMR\amr-main\explore\pathogen_paper\get_burden_estimates_draws.py

```python

import re
import os
import getpass
import glob
import argparse
import pandas as pd
import numpy as np
from pathlib import Path
import importlib
import ipdb
from datetime import date
from get_draws.api import get_draws

from db_queries import get_cause_metadata
from cod_prep.downloaders import get_pop, get_current_location_hierarchy
from cod_prep.claude.configurator import Configurator
import db_queries.api.public as db

from cod_prep.utils import print_log_message
from cod_prep.downloaders import get_ages
from amr_prep.utils.amr_io import get_amr_results
from mcod_prep.utils.mcause_io import get_mcause_results
from db_queries import get_outputs as go
from db_queries import get_population

# directories

DIR = 'FILEPATH'
repo_dir = 'FILEPATH'
MAP_DIR = 'FILEPATH'

DRAWS_DIR = 'FILEPATH'

CONF = Configurator('standard')
LSV_ID = CONF.get_id('location_set_version')
lh = get_current_location_hierarchy(location_set_version_id=LSV_ID)
ch = get_cause_metadata(cause_set_id=3, gbd_round_id=6, decomp_step="step4")

locs = lh.loc[lh["level"].isin([1,3]), "location_id"].unique().tolist() + [1]
ages = list(range(1,21))+[30, 31, 32, 235, 22, 27]
measures = [1,4]
metrics = [1,2]
sexes = [3]


gbd_unadj_causes_ids =[1027, 349, 843, 398, 405, 354, 355, 356, 397, 362, 363, 364, 936,
408]
gbd_adjusted_fpath = f'{DIR}gbd_causes_adjusted.csv'
gbd_adjusted_draws = f'{DIR}gbd_causes_adjusted_draws.csv'

pops = get_pop(release_id=6, pop_run_id=192)
```

```
pops = pops[pops.location_id.isin(locs)]

stomach_cancer_path= (f'{DIR}/stomach_cancer_adj.csv')
stomach_cancer_draws = (f'{DIR}/stomach_cancer_adj_draws.csv')

def pull_amr_causes_draws_u5():

    df_ftl_u5_raw = get_amr_results(
    process="summarize_burden",
    burden_type='fatal',
    draws=True,
    year_id=2019,
    cause_id=294,
    age_group_id=[2,3,4,5],
    location_id = locs,
    measure_id=[1],
    metric_id=[1,2], #number
    sex_id=3,
    hosp="all"
    )

    df_nonftl_u5_raw = get_amr_results(
    process="summarize_burden",
    burden_type='nonfatal',
    draws=True,
    year_id=2019,
    cause_id=294,
    age_group_id=[2,3,4,5],
    location_id = locs,
    measure_id=[2],
    metric_id=[1,2], #number
    sex_id=3,
    hosp="all"
    )

    return pd.concat([df_ftl_u5_raw, df_nonftl_u5_raw])

def pull_amr_causes_draws_all_ages():

    df_ftl_aa_raw = get_amr_results(
    process="summarize_burden",
    burden_type='fatal',
    draws=True,
    year_id=2019,
    cause_id=294,
    age_group_id=[22],
    location_id = locs,
    measure_id=[1],
    metric_id=[1,2], #number
    sex_id=3,
    hosp="all"
    )

    df_nonftl_aa_raw = get_amr_results(
```

```
    process="summarize_burden",
    burden_type='nonfatal',
    draws=True,
    year_id=2019,
    cause_id=294,
    age_group_id=[22],
    location_id = locs,
    measure_id=[2],
    metric_id=[1,2],
    sex_id=3,
    hosp="all"
    )

    return pd.concat([df_ftl_aa_raw, df_nonftl_aa_raw])

def get_additional_gbd_causes_draws(gbd_causes=gbd_unadj_causes_ids):

    '''
    Get draws for all gbd unadjusted causes, calculate counts/rates for Deaths, calucalte count/rates DALYs

    parameters:
    gbd_casuse - a list of gbd casuses that don't need additional chain/UCoD adjustment

    returns:
    a dataframe of 1000 draws, for Deaths and DALYs, counts and rates, all ages und under 5 age groups
    '''


    def check_metric_ids(df):
        '''
        check the both rates and counts metrics are present
        '''
        assert 1 in df.metric_id.unique(), 'no counts are present'
        assert 3 in df.metric_id.unique(), 'no rates are present'

    # get dalys rates

    cc = get_draws("cause_id",gbd_causes,
                source="codcorrect",
                metric_id=[1],
                measure_id=[1,4],
                release_id=6,
                location_id=locs,
                sex_id=3,
                year_id=2019,
                age_group_id=[1,22])

    dn = get_draws("cause_id",gbd_causes,
                        source="dalynator",
                        metric_id=[1],
                        release_id=6,
                        location_id=locs,
```

```
                        sex_id=3,
                        year_id=2019,
                        age_group_id=[1,22])


    pops = get_population(age_group_id=[1, 22], location_id=locs, year_id=2019, sex_id=3, release_id=6)
    draw_cols = [f'draw_{i}' for i in range(1000)]

    # Get DALYs rates
    dn = dn.merge(pops[["age_group_id", "location_id", "population"]], how = 'left', on=["age_group_id", "location_id"], validate='m:1')
    dn_rate = dn.copy()
    for col in draw_cols:
        dn_rate[col] = dn_rate[col]/dn_rate['population']
    dn_rate['metric_id'] = 3
    dalys = pd.concat([dn, dn_rate])
    dalys = dalys.drop(['population','version_id'], axis=1)

    assert len(dn.cause_id.unique())==14, 'not the correct number of gbd un-adjusted causes'
    assert 1 in dn.age_group_id.unique()
    assert 22 in dn.age_group_id.unique()

    # Get Deaths rates
    cc = cc.merge(pops[["age_group_id", "location_id", "population"]], how = 'left', on=["age_group_id", "location_id"], validate='m:1')
    cc_rate = cc.copy()
    for col in draw_cols:
        cc_rate[col] = cc_rate[col]/cc_rate['population']
    cc_rate['metric_id'] = 3
    cc = pd.concat([cc, cc_rate])
    cc = cc.drop(['population','version_id'], axis=1)
    deaths = cc[cc.measure_id==1]

    assert 1 in deaths.age_group_id.unique()
    assert 22 in deaths.age_group_id.unique()

    check_metric_ids(deaths)
    check_metric_ids(dalys)

    return pd.concat([deaths, dalys])


def add_viral_meningitis(df):

    '''
    Add AMR estimates for viral meningitis
    '''

    v_m = df.loc[(df.infectious_syndrome=='cns_infectious')&(df.pathogen=='virus')]
    v_m['pathogen'] = 'viral_meningitis'

    # filter "all" infectious syndromes
```

```
    df = df.loc[df.infectious_syndrome=='all']

    # append the viral meningitis
    df = pd.concat([df,v_m])
    # remove not needed columns
    df.drop(columns=['infectious_syndrome', 'abx_class', 'abx_set','counterfactual',
'cause_id','hosp'], inplace=True)
    return df

def add_pathogen_name_amr(df):

    '''update pathogen name for the AMR causes based on the pathogen_metadata. It is not the
final step in assigning pathogen names for the results
    '''
    pathogen_mdata = pd.read_csv(f'FILEPATH')

    df = df.merge(pathogen_mdata[['pathogen','pathogen_name_long']],how= 'left')
    df.rename(columns = {'pathogen_name_long':'pathogen_name'}, inplace=True)

    df.loc[df.pathogen=='viral_meningitis', 'pathogen_name'] = 'Viral meningitis'
    df['pathogen'] = df.pathogen_name
    df = df.drop(['pathogen_name'], axis=1)

    return df

def add_rates(df, pops_df=pops):

    df = df.merge(pops_df, how='left', on =
['age_group_id','location_id','year_id','sex_id'], validate='m:1')

    draw_cols = [f'draw_{i}' for i in range(1000)]
    df_rate = df.copy()

    for col in draw_cols:
        df_rate[col] = df_rate[col]/df_rate['population']
    df_rate['metric_id'] = 3

    df =pd.concat([df, df_rate])
    df = df.drop('population',axis=1)

    return df

def format_pathogens_in_final_results(df):

    '''
    Format pathogen names to meet the requirements for the results

    '''

    ptgns = pd.read_csv(f'FILEPATH')

    ptgns = ptgns[['pathogen_name','pathogen_name_updated']]
    ptgn_list = list(ptgns.pathogen_name_updated.unique())
    df = df.merge(ptgns, how='left', left_on='pathogen', right_on='pathogen_name')
```

```
260
261     df.pathogen = df['pathogen_name_updated']
262     df = df.sort_values(by='pathogen')
263
264     return df
265
266 def format_results(df, places = lh[['location_id','location_name']]):
267
268     ages = get_ages()
269     measure_dict = {1:'Deaths', 2: 'DALYs',  3: 'YLDs', 4: 'YLLs'}
270     metric_dict = {1:'Count', 3: 'Rate'}
271
272     df['measure'] = df.measure_id.map(measure_dict)
273     df['metric'] = df.metric_id.map(metric_dict)
274     df = df.merge(ages[['age_group_id','age_group_name']], how='left', on='age_group_id',
    validate='m:1')
275     df = df.merge(places, how='left', on='location_id', validate='m:1')
276
277     return df
278
279 def output_global_results(df,age_id, measure, metric):
280
281
282     cols = ['location_name','age_group_id','age_group_name','pathogen','mean_val',
    'upper_val', 'lower_val','source','measure','metric']
283     df = df.loc[(df.measure==measure)&(df.metric==metric)&(df.age_group_id==age_id),cols]
284     df = df.sort_values(by='pathogen')
285     age_name = df.age_group_name.unique()[0]
286
287     return df
288
289 def save_global_results(df, dir_to_save, age_g):
290
291     assert age_g in ['Under_5', 'All_Ages'], "use 'Under_5', 'All_Ages'"
292     age_dict = {'Under_5':1 , 'All_Ages':22}
293     age_id = age_dict[age_g]
294     today = date.today()
295     file_suffix = today.strftime("%m%d%y")
296
297     global_df = df.loc[df.location_name=='Global']
298     dalys_count = output_global_results(global_df, age_id, 'DALYs','Count')
299     deaths_count = output_global_results(global_df,age_id, 'Deaths','Count')
300     dalys_rate = output_global_results(global_df,age_id, 'DALYs','Rate_per_100_000')
301     deaths_rate = output_global_results(global_df,age_id, 'Deaths','Rate_per_100_000')
302
303
304     with pd.ExcelWriter(f'{dir_to_save}global_{age_g}_burden_estimates{file_suffix}.xlsx')
    as writer:
305
306         # use to_excel function and specify the sheet_name and index
307         # to store the dataframe in specified sheet
308         dalys_count.to_excel(writer, sheet_name=f"DALYs_Count_{age_g}", index=False,
    float_format='%10.2f')
```

```
        deaths_count .to_excel(writer, sheet_name=f"Deaths_Count_{age_g}", index=False, float_format='%10.2f')
        dalys_rate.to_excel(writer, sheet_name=f"DALYs_Rate_{age_g}", index=False, float_format='%10.2f')
        deaths_rate.to_excel(writer, sheet_name=f"Deaths_Rate_{age_g}", index=False, float_format='%10.2f')


def output_regional_results(df, age_id, measure, metric):

    cols = ['location_name','age_group_id','age_group_name','pathogen','mean_val', 'upper_val', 'lower_val','source','measure','metric']
    df = df.loc[(df.measure==measure)&(df.metric==metric)&(df.age_group_id==age_id),cols]


    df.rename(columns={'upper_val':f'{measure} {metric} upper',
                       'lower_val': f'{measure} {metric} lower'},inplace=True)
    df = df.sort_values(by='pathogen')
    df = df.pivot(index ='pathogen', columns='location_name', values=['mean_val', f'{measure} {metric} lower', f'{measure} {metric} upper'])

    return df

def save_regional_results(df, dir_to_save, age_g):

    '''
    output all the results by regions
    argunemts:
        df: results dataframe
        dir_to_save: the output directory
        age_g: age group

    '''

    today = date.today()
    file_suffix = today.strftime("%m%d%y")

    assert age_g in ['Under_5', 'All_Ages'], "use 'Under_5', 'All_Ages'"
    age_dict = {'Under_5':1 , 'All_Ages':22}
    age_id = age_dict[age_g]

    region_ids = lh.loc[lh.level==1, 'location_id'].unique()
    region_results = df.loc[df.location_id.isin(region_ids)]

    df1 = output_regional_results(region_results, age_id, 'DALYs','Count')
    df2 = output_regional_results(region_results, age_id, 'Deaths','Count')
    df3 = output_regional_results(region_results, age_id, 'DALYs','Rate_per_100_000')
    df4 = output_regional_results(region_results, age_id,  'Deaths','Rate_per_100_000')


    with pd.ExcelWriter(f'{dir_to_save}regional_{age_g}_burden_estimates_{file_suffix}.xlsx') as writer:

```

```
        # use to_excel function and specify the sheet_name and index
        # to store the dataframe in specified sheet
        df1.to_excel(writer, sheet_name=f"DALYs_Count_{age_g}", float_format='%10.3f')
        df2.to_excel(writer, sheet_name=f"Deaths_Count_{age_g}", float_format='%10.3f')
        df3.to_excel(writer, sheet_name=f"DALYs_Rate_{age_g}", float_format='%10.3f')
        df4.to_excel(writer, sheet_name=f"Deaths_Rate_{age_g}", float_format='%10.3f')



if __name__ == '__main__':

    gbd = get_additional_gbd_causes_draws()

    gbd_adj = pd.read_csv(gbd_adjusted_draws)

    gbd_adj['age_group_id'] = gbd_adj.age_group_name.map({'All Ages':22, 'Under 5':1 })

    amr_aa = pull_amr_causes_draws_all_ages()
    amr_u5 = pull_amr_causes_draws_u5()

    sc = pd.read_csv(stomach_cancer_draws)

    # get the list of amr pathogens we estimate
    all_amr_pathogens = amr_aa.loc[amr_aa.infectious_syndrome=='all', 'pathogen'].unique()

    exclude_pathogens =
['(none_estimated)','neisseria_gonorrheae','all','virus','other','mycobacterium_tuberc↩
ulosis']
    amr_pathogens = [i for i in all_amr_pathogens if i not in exclude_pathogens] +
['viral_meningitis']
    amr_pathogens.sort()

    amr_aa = add_viral_meningitis(amr_aa)
    amr_u5 = add_viral_meningitis(amr_u5)

    # filter by pathogens

    amr_aa = amr_aa.loc[amr_aa.pathogen.isin(amr_pathogens)]
    amr_u5 = amr_u5.loc[amr_u5.pathogen.isin(amr_pathogens)]

    # aggregate under 5

    amr_u5_agg = amr_u5.groupby([ 'location_id','pathogen','sex_id',
'measure_id','year_id','metric_id']).sum().reset_index()
    amr_u5_agg['age_group_id'] = 1

    amr = pd.concat([amr_aa, amr_u5_agg])
    amr = add_pathogen_name_amr(amr)

    gbd['source'] = 'gbd'
    amr['source'] = 'amr'
    gbd_adj['source'] = 'gbd_adjusted'

    gbd_adj['metric_id'] = 1
```

```
    gbd_adj['year_id'] = 2019
    gbd_adj['sex_id'] = 3

    # keep only DALYs and Deaths
    gbd_adj = gbd_adj[gbd_adj.measure_id.isin([1,2])]

    sc['source'] = 'stomach_cancer'
    sc['year_id'] = 2019

    gbd = gbd.merge(ch[['cause_id','cause_name']], how='left', on='cause_id',
validate='m:1')
    gbd = gbd.rename(columns={'cause_name':'pathogen'})

    measure_dict = {1:'Deaths',
                    2: 'DALYs',
                    3: 'YLDs',
                    4: 'YLLs'}

    metric_dict = {1:'Count',
                   3: 'Rate'}


    # add rates
    gbd_adj = add_rates(gbd_adj)
    amr = add_rates(amr)
    sc = add_rates(sc)

    results = pd.concat([gbd, gbd_adj, amr, sc])
    results = results.drop(['cause_id','age_group_name'], axis=1)

    # update the pathogen names based on the needs of the paper
    results = format_pathogens_in_final_results(results)
    results = results.drop(['pathogen', 'pathogen_name'], axis=1)
    results = results.rename(columns={'pathogen_name_updated':'pathogen'})

    draw_cols = [f'draw_{i}' for i in range(0,1000)]
    results['mean_val'] = results[draw_cols].apply(np.mean, axis=1)
    results['lower_val'] = results[draw_cols].apply(np.quantile,q=0.025, axis=1)
    results['upper_val'] = results[draw_cols].apply(np.quantile,q=0.975, axis=1)

    cols_to_keep = ['metric_id', 'measure_id', 'age_group_id','location_id', 'sex_id',
'year_id', 'source', 'pathogen', 'mean_val', 'lower_val','upper_val']

    locs = lh.loc[lh["level"].isin([1]), "location_id"].unique().tolist()+[1]

    results_rg = results.loc[results.location_id.isin(locs)]

    cols_to_agg = ['metric_id', 'measure_id', 'age_group_id','location_id', 'sex_id',
'year_id']
    results_rg = results_rg.groupby(cols_to_agg).sum().reset_index(0)


    results_rg['mean_val'] = results_rg[draw_cols].apply(np.mean, axis=1)
    results_rg['lower_val'] = results_rg[draw_cols].apply(np.quantile,q=0.025, axis=1)
```

```
    results_rg['upper_val'] = results_rg[draw_cols].apply(np.quantile,q=0.975, axis=1)

    totals = results_rg.reset_index()
    totals['pathogen'] = 'total'

    results = pd.concat([results, totals])

    # remove draw_cols
    results = results.drop(draw_cols, axis=1)

    # add measure, metric, age_group_name, location_name (def format_results)
    results = format_results(results)

    # add rates per 100_000
    results_rate = results[results.metric_id==3]

    for col in ['mean_val','lower_val','upper_val']:
        results_rate[col] = results_rate[col]*100_000

    results_rate['metric'] = 'Rate_per_100_000'
    results_rate['metric_id'] = np.nan
    results = pd.concat([results, results_rate])

    # save all results in default format
    results.to_csv(f'{DIR}FILEPATH', index=False)

    output_dir = DIR
    age_string = 'All_Ages'
    save_global_results(results, output_dir, age_string)

    age_string = 'Under_5'
    save_global_results(results, output_dir, age_string)

    output_dir = DIR
    age_string = 'All_Ages'
    save_regional_results(results, output_dir, age_string)

    age_string = 'Under_5'
    save_regional_results(results, output_dir, age_string)
```