D:\NHRI\AMR\amr-main\explore\number_plug_amr_results.py

```python
import pandas as pd
import numpy as np
import seaborn as sns
from pathlib import Path
from db_tools import ezfuncs
from cod_prep.downloaders import *
from cod_prep.utils import *
from cod_prep.claude.configurator import Configurator
from amr_prep.utils.amr_io import get_amr_results
from amr_prep.utils.amr_io import AmrResult
from mcod_prep.compile_burden import summarize_draws
import warnings
warnings.filterwarnings("ignore")

CONF = Configurator()
LSV_ID = CONF.get_id('location_set_version')
CSV_ID = CONF.get_id('cause_set_version')

lh = get_current_location_hierarchy(location_set_version_id=LSV_ID)
ch = get_current_cause_hierarchy(cause_set_version_id=CSV_ID)


def get_results_wrapper(burden, measure, metric, syndrome=None,
                        pathogen=None, abx_class=None, draws=False,
                        location_id=1, counterfactual=None):
    df = get_amr_results(
        'summarize_burden',
        burden,
        year_id=2019,
        cause_id=294,
        location_id=location_id,
        age_group_id=22,
        infectious_syndrome=syndrome,
        pathogen=pathogen,
        abx_class=abx_class,
        counterfactual=counterfactual,
        measure_id=measure,
        metric_id=metric,
        sex_id=3,
        hosp='all',
        draws=draws,
        filter_continuously=True
    )
    return df


def print_out_cf_lower_mean_upper(df, measure):
    counterfactuals = df['counterfactual'].unique().tolist()

    for counterfactual in counterfactuals:

```

```
        lower = df.loc[df['counterfactual'] == counterfactual, 'amr_lower'].item()
        mean = df.loc[df['counterfactual'] == counterfactual, 'amr_mean'].item()
        upper = df.loc[df['counterfactual'] == counterfactual, 'amr_upper'].item()

        if counterfactual == 'no_infection':
            counterfactual = 'associated'
        elif counterfactual == 'susceptible_infection':
            counterfactual = 'attributable'

        print(counterfactual + ' ' + measure + ': '
            + str(mean) + ' (' + (str(lower) + ' - '+ str(upper)) + ')')


def aggregate_summarize_draws(df, to_aggregate, get_pct=False, denominator=None):
    if isinstance(to_aggregate, str):
        to_aggregate = [to_aggregate]
    df.drop(columns=to_aggregate, inplace=True)
    non_draw_cols = [col for col in df.columns if 'draw_' not in col]
    draw_cols = [col for col in df.columns if 'draw_' in col]
    df = df.groupby(non_draw_cols, as_index=False)[draw_cols].sum()

    if get_pct:
        assert (denominator is not None), 'If you would like to calculate pct, please pass
in denominator'
        denominator.drop(columns=to_aggregate, inplace=True)
        df = df.set_index(non_draw_cols)
        denominator = denominator.set_index(non_draw_cols)
        df = df.reorder_levels(non_draw_cols)
        denominator = denominator.reorder_levels(non_draw_cols)
        assert len(df) == len(denominator)
        pct = df.div(denominator)
        pct = pct.reset_index()

        dfs = []
        for adf in [df, pct]:
            adf = adf.reset_index()
            adf = summarize_draws(adf, draw_cols=draw_cols, prefix='amr_')
            adf = adf.reset_index(drop=True)
            adf.loc[
                adf.counterfactual == 'susceptible_infection',
                ['amr_mean', 'amr_upper', 'amr_lower']
            ] = np.clip(adf[['amr_mean', 'amr_upper', 'amr_lower']], 0, None)
            assert adf.notnull().values.all()
            dfs.append(adf)
        return dfs[0], dfs[1]
    else:
        df = summarize_draws(df, draw_cols=draw_cols, prefix='amr_')
        df = df.reset_index(drop=True)
        df.loc[
            df.counterfactual == 'susceptible_infection',
            ['amr_mean', 'amr_upper', 'amr_lower']
        ] = np.clip(df[['amr_mean', 'amr_upper', 'amr_lower']], 0, None)
        assert df.notnull().values.all()
        return df
```

```
print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')

for burden in ['fatal', 'nonfatal']:
    print()
    print('--------------------------' + burden.upper() + '----------------------------')
    if burden == 'fatal':
        measure='Deaths'
        measure_id=1
    elif burden == 'nonfatal':
        measure='DALYs'
        measure_id=2

    print('Combo counts for fatal and nonfatal')
    df = get_results_wrapper(burden, measure_id, 1, syndrome='all')
    df = df.loc[~df['pathogen'].isin(['all', '(none_estimated)']), ]
    df = df.loc[df['abx_class'] != '(none_tested)', ]
    df.loc[df['pathogen'].str.contains('_escherichia'), 'pathogen'] = 'escherichia_coli'
    print(burden + ' has ' + str(len(df[['pathogen']].drop_duplicates())) + ' pathogens')
    print(df['pathogen'].unique().tolist())
    df = df.loc[~df['abx_class'].str.contains('all_'), ]
    print(burden + ' has ' + str(len(df[['pathogen', 'abx_class']].drop_duplicates())) + ' combos')
    print(df[['pathogen', 'abx_class']].drop_duplicates())

    print()
    print('Total')
    df = get_results_wrapper(burden, measure_id, 1, syndrome='all', pathogen='all', abx_class='all_resistant')
    print('Total AMR ' + measure + ':')
    print_out_cf_lower_mean_upper(df, measure)

    print()
    print('Top 3 syndromes')
    df = get_results_wrapper(burden, measure_id, 1, pathogen='all', abx_class='all_resistant')
    both_cfs = []
    for counterfactual in ['no_infection', 'susceptible_infection']:
        print(counterfactual)
        print(df.loc[(df['infectious_syndrome'] != 'all')
                   & (df['counterfactual'] == counterfactual),
            ['infectious_syndrome', 'pathogen', 'amr_lower', 'amr_mean', 'amr_upper']]
            .sort_values(by='amr_mean', ascending=False))
        top3 = df.loc[(df['infectious_syndrome'] != 'all')
                     & (df['counterfactual'] == counterfactual),
            ].sort_values(by='amr_mean', ascending=False).head(3)['infectious_syndrome'].unique().tolist()
        both_cfs.append(top3)
    assert set(both_cfs[0]) == set(both_cfs[1])
    df = get_results_wrapper(burden, measure_id, 1, syndrome=both_cfs[0], pathogen='all',
        abx_class='all_resistant', draws=True)
    total = get_results_wrapper(burden, measure_id, 1,
                                syndrome='all', pathogen='all', abx_class='all_resistant',
```

```
                                draws=True)
    df, pct = aggregate_summarize_draws(df, to_aggregate='infectious_syndrome', get_pct=True, denominator=total)
    print('Top 3 syndromes AMR ' + measure + ':')
    print_out_cf_lower_mean_upper(df, measure)
    print('Top 3 syndromes AMR percentage out of total ' + measure + ':')
    print_out_cf_lower_mean_upper(pct, measure)

    print()
    print('Top 6 pathogens')
    df = get_results_wrapper(burden, measure_id, 1, syndrome='all', abx_class='all_resistant')
    both_cfs = []
    for counterfactual in ['no_infection', 'susceptible_infection']:
        print(counterfactual)
        print(df.loc[(df['pathogen'] != 'all') & (df['counterfactual'] == counterfactual),
            ['infectious_syndrome', 'pathogen', 'amr_lower', 'amr_mean', 'amr_upper']]
            .sort_values(by='amr_mean', ascending=False))
        top6 = df.loc[(df['pathogen'] != 'all') & (df['counterfactual'] == counterfactual),
            ].sort_values(by='amr_mean', ascending=False).head(6)['pathogen'].unique().tolist()
        both_cfs.append(top6)
    df = get_results_wrapper(burden, measure_id, 1, syndrome='all', pathogen=both_cfs[0],
        abx_class='all_resistant', draws=True)
    total = get_results_wrapper(burden, measure_id, 1,
                                syndrome='all', pathogen='all', abx_class='all_resistant',
                                draws=True)
    df, pct = aggregate_summarize_draws(df, to_aggregate='pathogen', get_pct=True, denominator=total)
    print('Top 6 pathogens AMR ' + measure + ':')
    print_out_cf_lower_mean_upper(df, measure)
    print('Top 6 pathogens AMR percentage out of total ' + measure + ':')
    print_out_cf_lower_mean_upper(pct, measure)

    print()
    print('Top Combos')
    df = get_results_wrapper(burden, measure_id, 1, syndrome='all')
    both_cfs = []
    for counterfactual in ['no_infection', 'susceptible_infection']:
        print(counterfactual)
        print(df.loc[(df['pathogen'] != 'all')
            & (df['counterfactual'] == counterfactual)
            & ~(df['abx_class'].str.contains('all_'))
            & (df['abx_class'] != '(none_tested)'),
            ['infectious_syndrome', 'pathogen', 'abx_class', 'amr_lower', 'amr_mean', 'amr_upper']]
                .sort_values(by='amr_mean', ascending=False)
                .head(20))

    print()
    print('Rate By Super Regions and Regions')
    regions = lh.loc[lh['level'] <= 2, 'location_id'].unique().tolist()
    df = get_results_wrapper(burden, measure_id, 3,
        syndrome='all', pathogen=['all'],
```

```
        abx_class=['all_resistant'], location_id=regions)
    df = add_location_metadata(df, 'location_name')
    df.loc[df.metric_id == 3, ['amr_mean', 'amr_lower', 'amr_upper']] = \
        df[['amr_mean', 'amr_lower', 'amr_upper']] * 100_000
    both_cfs = []
    for counterfactual in ['no_infection', 'susceptible_infection']:
        print(counterfactual)
        pd.set_option('display.max_rows', 600)
        print(df.loc[(df['counterfactual'] == counterfactual),
            ['location_name', 'infectious_syndrome', 'pathogen', 'amr_lower', 'amr_mean', 'amr_upper']]
                .sort_values(by='amr_mean', ascending=False))
    print()


print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')


print('MISC One-offs')
print()
print('Find S. Aureus, E. Coli in High-Income')
df = get_results_wrapper('fatal', 1, 1,
    pathogen=['staphylococcus_aureus'], syndrome='all', abx_class='all_resistant', location_id=64,
    draws=True)
total = get_results_wrapper('fatal', 1, 1,
    syndrome='all', pathogen='all', abx_class='all_resistant', location_id=64, draws=True)
df, pct = aggregate_summarize_draws(df, to_aggregate='pathogen', get_pct=True, denominator=total)
print('S. Aureus AMR percentage out of total in High-Income deaths:')
print_out_cf_lower_mean_upper(pct, 'deaths')


df = get_results_wrapper('fatal', 1, 1,
    pathogen=['escherichia_coli'], syndrome='all', abx_class='all_resistant', location_id=64,
    draws=True)
total = get_results_wrapper('fatal', 1, 1,
    syndrome='all', pathogen='all', abx_class='all_resistant', location_id=64, draws=True)
df, pct = aggregate_summarize_draws(df, to_aggregate='pathogen', get_pct=True, denominator=total)
print('E. Coli AMR percentage out of total in High-Income deaths:')
print_out_cf_lower_mean_upper(pct, 'deaths')

print()
print('Find S. pneumoniae, K. pneumoniae in sub-Saharan Africa')
df = get_results_wrapper('fatal', 1, 1,
    pathogen=['streptococcus_pneumoniae'], syndrome='all', abx_class='all_resistant', location_id=166,
    draws=True)
total = get_results_wrapper('fatal', 1, 1,
    syndrome='all', pathogen='all', abx_class='all_resistant', location_id=166, draws=True)
df, pct = aggregate_summarize_draws(df, to_aggregate='pathogen', get_pct=True, denominator=total)
```

```
print('S. pneumoniae AMR percentage out of total in sub-Saharan Africa deaths:')
print_out_cf_lower_mean_upper(pct, 'deaths')


df = get_results_wrapper('fatal', 1, 1,
    pathogen=['klebsiella_pneumoniae'], syndrome='all', abx_class='all_resistant', location_id=166,
    draws=True)
total = get_results_wrapper('fatal', 1, 1,
    syndrome='all', pathogen='all', abx_class='all_resistant', location_id=166, draws=True)
df, pct = aggregate_summarize_draws(df, to_aggregate='pathogen', get_pct=True, denominator=total)
print('K. pneumoniae AMR percentage deaths out of total in sub-Saharan Africa:')
print_out_cf_lower_mean_upper(pct, 'deaths')


print()
print('Check that fqns and beta lactams account over 70% burden attributable')
abxs = [
    'anti_pseudomonal_penicillin',
    'carbapenem',
    'fourth_gen_ceph',
    'third_gen_ceph',
    'fluoroquinolone',
    'aminopenicillin',
    'penicillin',
    'methicillin',
    'beta_lactamase_inhibitor'
]

df = get_results_wrapper('fatal', 1, 1,
    syndrome='all', pathogen='all', abx_class=abxs, counterfactual='susceptible_infection', draws=True)
print('check... ' + str(df['abx_class'].unique().tolist()))
total = get_results_wrapper('fatal', 1, 1,
    syndrome='all', pathogen='all', abx_class='all_resistant', counterfactual='susceptible_infection', draws=True)
df, pct = aggregate_summarize_draws(df, to_aggregate='abx_class', get_pct=True, denominator=total)
print('flqs and beta lactams total attributable deaths out of total in global:')
print_out_cf_lower_mean_upper(pct, 'deaths')


overlap_4 = [
    'mycobacterium_tuberculosis',
    'staphylococcus_aureus',
    'escherichia_coli',
    'klebsiella_pneumoniae'
]
df = get_results_wrapper('fatal', 1, 1, syndrome='all', pathogen=overlap_4,
    counterfactual='susceptible_infection', draws=True, location_id=[42, 73])
df = aggregate_summarize_draws(df, to_aggregate=['pathogen', 'abx_class', 'location_id'])
print_out_cf_lower_mean_upper(df, 'deaths')

```

```
300
301 print()
302
303 print('Find the total burden attributable for the 11 combos that overlap with cassini')
304
305 cassini_overlap = [
306     ['carbapenem', 'acinetobacter_baumanii'],
307     ['vancomycin', 'enterococcus_faecalis'],
308     ['vancomycin', 'enterococcus_faecium'],
309     ['carbapenem', 'escherichia_coli'],
310     ['third_gen_ceph', 'escherichia_coli'],
311     ['carbapenem', 'klebsiella_pneumoniae'],
312     ['third_gen_ceph', 'klebsiella_pneumoniae'],
313     ['carbapenem', 'pseudomonas_aeruginosa'],
314     ['methicillin', 'staphylococcus_aureus'],
315     ['penicillin', 'streptococcus_pneumoniae'],
316     ['macrolide', 'streptococcus_pneumoniae'],
317 ]
318
319 overlap_pathogens =  [
320     'acinetobacter_baumanii',
321     'enterococcus_faecalis',
322     'enterococcus_faecium',
323     'escherichia_coli',
324     'klebsiella_pneumoniae',
325     'pseudomonas_aeruginosa',
326     'staphylococcus_aureus',
327     'streptococcus_pneumoniae',
328 ]
329
330 df = get_results_wrapper('fatal', 1, 1, syndrome='all', pathogen=overlap_pathogens,
331     counterfactual='susceptible_infection', draws=True, location_id=[42, 73])
332
333 dfs = []
334 for combo in cassini_overlap:
335     adf = df.loc[(df['abx_class'] == combo[0]) & (df['pathogen'] == combo[1]), ]
336     dfs.append(adf)
337 df = pd.concat(dfs)
338 print('Total deaths attributable for 11 combos overlap with Cassini in Central and Western
    Europe')
339
340 df = aggregate_summarize_draws(df, to_aggregate=['pathogen', 'abx_class', 'location_id'])
341 print_out_cf_lower_mean_upper(df, 'deaths')
342
343 df = get_results_wrapper('nonfatal', 2, 1, syndrome='all', pathogen=overlap_pathogens,
344     counterfactual='susceptible_infection', draws=True, location_id=[42, 73])
345
346 dfs = []
347 for combo in cassini_overlap:
348     adf = df.loc[(df['abx_class'] == combo[0]) & (df['pathogen'] == combo[1]), ]
349     dfs.append(adf)
350 df = pd.concat(dfs)
351 print('Total DALYs attributable for 11 combos overlap with Cassini in Central and Western
    Europe')
```

```
df = aggregate_summarize_draws(df, to_aggregate=['pathogen', 'abx_class', 'location_id'])
print_out_cf_lower_mean_upper(df, 'DALYs')


print()
print('check cardiac compared to total associated')
df = get_results_wrapper('fatal', 1, 1, syndrome='cardiac_infectious', pathogen='all',
abx_class='all_resistant',
    counterfactual='no_infection', draws=True)
total = get_results_wrapper('fatal', 1, 1, syndrome='all', pathogen='all',
abx_class='all_resistant',
    counterfactual='no_infection', draws=True)
df, pct = aggregate_summarize_draws(df, to_aggregate='infectious_syndrome', get_pct=True,
denominator=total)
print_out_cf_lower_mean_upper(pct, 'deaths')

```