Supplementary Appendices

Supplementary Appendix 1

Household method logic coded in Python

```
#!/usr/bin/env python3

# new_pae.py
# Script takes 1 parameter: event_date (date to define place of residence )
# eg python3 new_pae.py '2012-01-01'

# Import libraries/modules
import pymssql  # interacting with Microsoft SQL Server
import pandas as pd  # data manipulation and analysis
import configparser  # reading configuration files
from datetime import datetime  # datetime class from the datetime module
import sys

##############################################################################
#############################
#           FUNCTIONS
##############################################################################
#############################

#### function DH() converts a date string ("YYYY-MM-DD") into a numeric (ret)
# runs faster than the pandas.to_datetime()
def DH(date):
    if (date == "None"): return 0  # Return 0 if the date is "None"

    # Split the date string into year, month, day and convert to integer
    z = date.split("-")
    year, month, day = z[0], z[1], z[2]
    y, m, d = int(year), int(month), int(day)

    # Calculate the number of leap years (r) and days (ret) since 1840 and days in February (leap)
    r = round((y - 1) // 4) - (round((y - 1) // 100)) + (round((y - 1) // 400)) - 446
    ret = 366 * r + ((y - 1841 - r) * 365) + d
    leap = 29 if (y % 4 >0) else 28 if (y % 100 >0) else 29 if (y % 400 >0) else 28

    # Add the number of days for each month (var) prior to the current month
    var = [31, leap, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    for i in range(len(var)):
        m = m - 1
        if (m == 0): break
        ret = ret + var[i]

    return ret  # Return the numerical representation of the date (ret)

#### function read_ini_extra() reads configuration from an INI file or a dictionary array.
# Extracts database connection parameters, NAME and DEBUG settings.
def read_ini_extra(file_path, dict_obj=None):
    global ZDB, ZUSER, ZPASS, ZHOST, ZPORT  # Declare global variables to store database connection parameters

    # Use ConfigParser() to read ini file or dictionary
    config = configparser.ConfigParser()
    if dict_obj:
        config.read_dict(dict_obj)
    else:
        config.read(file_path)
```

```
    # Get the value of DEBUG setting and convert it to boolean
    debug = config["APP"].getboolean("DEBUG")

    # Get the value of NAME setting
    name = config.get('APP', 'NAME', fallback='NAME is not defined')

    # Get the database connection parameters
    ZDB = config["DATABASE"].get("DB")  # Database name
    ZUSER = config["DATABASE"].get("USERNAME")  # Database username
    ZPASS = config["DATABASE"].get("PASSWORD")  # Database password
    ZHOST = config["DATABASE"].get("HOST")  # Database host
    ZPORT = config.get('DATABASE', 'PORT', fallback='PORT is not defined')  # Database port with a fallback value if not defined

    return debug  # Return the DEBUG setting (debug)

#### function GMSV3 determines the GMS registration status of a patient based on the event date.
# Checks if the patient exists and not deceased. Iterates through the patient's registration record (episodes of care) to determine their status.
def GMSV3(nor, event_date):
    b = 2
    zevent_date = DH(event_date) # Use DH() to convert event_date to numeric

    # Check if patient exists + if patient is deceased
    if (nor not in patient):
        return 4  # Patient not found

    dod = patient[nor][0] # Get date of death

    if (not str(dod) == "None"):
        dod_h = DH(str(dod))
        if (zevent_date >= dod_h):
            return b  # Patient is deceased

    # Iterate through the patient's episodes of care record
    # Check the episode is for a Regular/GMS registration (not dummy, emergency, temporary etc)
    # Check if the event date falls within or on the episode start and end dates + episode was active
    for i in x[nor]:
        z = i.split("~")
        id = z[0]
        date_start = z[1]
        date_end = z[2]
        type = z[3]

        d1 = DH(date_start)
        d2 = DH(date_end)

        if (type != "1335267"):
            continue # If registration type is not 1335267 (Regular/GMS patient), skip

        if (not d1==0 and d1 >zevent_date):
            continue # If the episode start date is after the event date, skip

        if (d1 < = zevent_date and d2 == 0):
            b = 1
            break # episode was active on the event date

        if (d2 >= zevent_date and d2 >= d1):
            b = 1
            break # event date falls within the episode date range
```

```
        if (d1 < zevent_date and d2 < d1):
            b = 3
            break # event date is after the end date of the episode

    return b  # Return a status code (b)

##################################################################################
#############################
#           FETCH PATIENT / ADDRESS DATA
##################################################################################
#############################
patient = {}; x = {}; adr = {}; adridx = {}; matchbig = {}  # Initialize dictionaries for storing data

# Connect to the database + set start time + clear the patient dictionary
ret = read_ini_extra("/tmp/bob.ini") # Use read_ini_extra() to get debug mode from INI file
conn = pymssql.connect(server=ZHOST, user=ZUSER, password=ZPASS, database=ZDB)
zstart = datetime.now()

########## patient
##################################################################################
##########
patient.clear()
# Execute SQL query to fetch patient data from database in batches of 1000000
cursor = conn.cursor()
cursor.execute('SELECT id, date_of_death FROM [compass_gp].[dbo].[patient] ORDER BY id OFFSET 3000001 ROWS FETCH
NEXT 1000000 ROWS ONLY;')

# Loop through each patient data record and add the patient ID and date of death into the patient dictionary
row = cursor.fetchone()
c = 1
while row:
    if (c % 10000 == 0):
        print(c);  # Print progress every 10000 rows
    id = row[0] # Patient ID
    date_of_death = row[1]
    patient[id] = [date_of_death]
    c = c + 1
    row = cursor.fetchone()
print(c)  # Print the total number of fetched rows

########## patient registration record (episode_of_care)
#################################
x.clear()
# Execute SQL query to fetch episode_of_care data from database into dictionary (x)
cursor = conn.cursor()
cursor.execute('SELECT id, patient_id, registration_type_concept_id, date_registered, date_registered_end FROM [compass_gp].[dbo
[episode_of_care];')

# Loop through each episode_of_care data record and add the episode_of_care ID, patient ID, registration type ID and
registration start/end dates for each patient
row = cursor.fetchone()
c = 1
while row:
    if (c % 10000 == 0):
        print(c)  # Print progress every 10000 rows
    id = row[0]  # Episode of care ID
    patient_id = row[1]
    type = row[2]
    date_start = row[3]
    date_end = row[4]
```

```
    if (patient_id in patient): # Check the episode of care patient ID matches with patient ID in the patient dictionary
        x.setdefault(patient_id, []).append(str(id) + "~" + str(date_start) + "~" + str(date_end) + "~" + str(type))  # Add
episode data to the dictionary
    c = c + 1
    row = cursor.fetchone()
print(c)  # Print the total number of fetched rows

########## patient address
############################################################################
adr.clear()
adridx.clear()
# Execute SQL query to fetch patient address data from database into dictionaries (adr) and (adridx)
cursor = conn.cursor()
cursor.execute('SELECT id, patient_id, start_date, end_date, use_concept_id, lsoa_2011_code, msoa_2011_code FROM
[compass_gp].[dbo].[patient_address];')

# Loop through each patient address record
# Add (for each patient) address ID to dictionary (adridx) and patient ID, residency start/end date, use ID, LSOA and MSOA, as
a list, to dictionary (adr)
row = cursor.fetchone()
c = 1
while row:
    if (c % 10000 == 0):
        print(c)  # Print progress every 10000 rows
    id = row[0]  # Address ID
    patient_id = row[1]
    start_date = row[2]
    end_date = row[3]
    use = row[4]
    lsoa = row[5]
    msoa = row[6]
    if (patient_id in patient): # Check the patient_address patient ID matches with patient ID in the patient dictionary
      adr.setdefault(patient_id, []).append([id, str(start_date), str(end_date), str(use), str(lsoa), str(msoa), DH(str(start_date))])
# Add to dictionary adr(list)
        adridx[id] = [] # Add to dictionary (adridx)
    c = c + 1
    row = cursor.fetchone()
print(c)  # Print the total number of fetched rows

########## patient address match
############################################################################
matchbig.clear()
# Execute SQL query to fetch patient address match data from database into dictionaries
cursor = conn.cursor()
cursor.execute('SELECT id, patient_address_id, uprn, qualifier, uprn_property_classification FROM [compass_gp].[dbo].
[patient_address_match];')

# Loop through each patient address match record
# Add (for each address) match ID, address ID, uprn, qualifier and classification to dictionary (matchbig)
row = cursor.fetchone()
c = 1
while row:
    if (c % 10000 == 0):
        print(c)  # Print progress every 10000 rows
    match_id = row[0]
    adr_id = row[1]
    if (adr_id not in adridx): # Check if the address ID exists in the adridx dictionary
        row = cursor.fetchone()
        continue
    uprn = row[2]
    qualifier = row[3]
```

```
    classification = row[4]
    if (adr_id in matchbig): # Update dictionary (matchbig) with the latest match information for each address ID
        m_id = matchbig[adr_id][0]
        if (match_id >m_id):
            matchbig[adr_id] = [match_id, uprn, qualifier, classification]
    else:
        matchbig[adr_id] = [match_id, uprn, qualifier, classification]
    c = c + 1
    row = cursor.fetchone()
print(c)  # Print the total number of fetched rows

##########################################################################
############################
#          MATCH PATIENT ADDRESS
##########################################################################
############################
event_date = str(sys.argv[1]) # takes input parameter
zevent_date = DH(event_date) # Use DH() to convert event_date into a numeric
uprn = ""

# Open a file for output and add the required headers
outputFile = open("/tmp/hh_output.txt", "w")
outputFile.write("patient_id\taddress_id\tuprn\taddress_start_date\taddress_end_date\taddress_use\tproperty_classification
\tlsoa\tmsoa\n")

# Define dictionary (VPROP) with property codes
VPROP = {"R": "", "RD": "", "RD01": "", "RD02": "", "RD03": "", "RD04": "", "RD06": "", "RD07": "", "RD10": "",
"RH02": "", "U": "", "UC": "",  "UP": "", "X": ""}
# R = Residential
# RD = Dwelling
# RD01 = Caravan
# RD02 = Detached
# RD03 = Semi-Detached
# RD04 = Terraced
# RD06 = Self Contained Flat (Includes Maisonette / Apartment)
# RD07 = House Boat
# RD10 = Privately Owned Holiday Caravan / Chalet
# RH02 = HMO Bedsit / Other Non Self Contained Accommodation
# U = Unclassified
# UC = Awaiting Classification
# UP = Pending Internal Investigation
# X = Dual Use

# Loop through each patient in the patient dictionary
c = 1  # Initialize a counter for progress tracking
for nor in patient:
    if (c % 10000 == 0):
        print(c)  # Print progress every 10000 patients
    c = c + 1
    ret = GMSV3(nor, event_date)# Use GMSV3() to determine the registration status of the patient
    if (ret == 2): # If the patient is dead (status code 2), write the patient ID and status to the output file
        outputFile.write(str(nor) + "\t2\n")
        continue
    if (nor in adr):# If the patient is in adr dictionary, sort addresses based on start date and address ID
        blist = sorted(adr[nor], key=lambda x: (x[6], x[0]), reverse=True)

        # SubLoop through each address associated with the patient
        # Extract patient_id, address ID, start date, end date, use concept ID, LSOA and MSOA into list (blist)
        for i in range(len(blist)):
            id = int(blist[i][0])
            start_date = blist[i][1]
```

```
        end_date = blist[i][2]
        use = blist[i][3]
        lsoa = blist[i][4]
        msoa = blist[i][5]

        #1335358 = Home
        #1335360 = Temporary
        #1335361 = Old / Incorrect
        if (use == "1335360"):# If address concept ID is 1335360 (temporary address), skip
            continue

        # If the address ID exists in matchbig dictionary, extract UPRN, qualifier, and classification
        if (id in matchbig):
            uprn = matchbig[id][1]
            qualifier = matchbig[id][2]
            classification = matchbig[id][3]

        if (uprn == ""): # If UPRN is empty, skip
            continue

        if (classification not in VPROP.keys()): # If the classification is not in VPROP dictionary keys, skip
            continue

        if (qualifier != "Best (residential) match"): # If the qualifier is not "Best (residential) match", skip
            continue

        d1 = DH(start_date)  # Convert start date to numeric
        d2 = DH(end_date)  # Convert end date to numeric

        # If the event date is within the date range of the address, write address details to the output file
        if (d1 <= zevent_date or d1 == 0) and (d2 >= zevent_date or d2 == 0):
            outputFile.write(str(nor) + "\t" + str(id) + "\t" + str(uprn) + "\t" + str(start_date) + "\t" + end_date + "\t"
            + use + "\t" + classification + "\t" + lsoa + "\t" + msoa + "\n")
            break  # Exit the SubLoop back into the main Loop

# Close the output file + Database Connection. Print run times
outputFile.close()
conn.close()
print(zstart)
print(datetime.now())
```

Supplementary Appendix 2: Proportions of the combinations of the four reasons a household RALF was not assigned to 10,025 children in the use case cohort

	Multiple different valid household UPRNs	No valid household UPRNs	No address records at event date	Not alive or no regular registrations at event dates	Total	%
0000	0	0	0	0	6	0.1
0001	0	0	0	1	3,091	30.8
0100	0	1	0	0	2,027	20.2
0101	0	1	0	1	3,506	35.0
0110	0	1	1	0	593	5.9
0111	0	1	1	1	379	3.8
1000	1	0	0	0	188	1.9
1001	1	0	0	1	229	2.3
1100	1	1	0	0	1	0.0
1101	1	1	0	1	5	0.0
Total					10,025	100

Note: the combinations are in binary format where 1 = any of the person's multiple address records met the criteria.