""" Module with functionalities for blocking based on a dictionary of records,

where a blocking function must return a dictionary with block identifiers

as keys and values being sets or lists of record identifiers in that block.

"""

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

def noBlocking(rec\_dict):

"""A function which does no blocking but simply puts all records from the

given dictionary into one block.

Parameter Description:

rec\_dict : Dictionary that holds the record identifiers as keys and

corresponding list of record values

"""

print("Run 'no' blocking:")

print(' Number of records to be blocked: '+str(len(rec\_dict)))

print('')

rec\_id\_list = list(rec\_dict.keys())

block\_dict = {'all\_rec':rec\_id\_list}

return block\_dict

# -----------------------------------------------------------------------------

def simpleBlocking(rec\_dict, blk\_attr\_list):

"""Build the blocking index data structure (dictionary) to store blocking

key values (BKV) as keys and the corresponding list of record identifiers.

A blocking is implemented that simply concatenates attribute values.

Parameter Description:

rec\_dict : Dictionary that holds the record identifiers as keys

and corresponding list of record values

blk\_attr\_list : List of blocking key attributes to use

This method returns a dictionary with blocking key values as its keys and

list of record identifiers as its values (one list for each block).

Examples:

If the blocking is based on 'postcode' then:

block\_dict = {'2000': [rec1\_id, rec2\_id, rec3\_id, ...],

'2600': [rec4\_id, rec5\_id, ...],

...

}

while if the blocking is based on 'postcode' and 'gender' then:

block\_dict = {'2000f': [rec1\_id, rec3\_id, ...],

'2000m': [rec2\_id, ...],

'2600f': [rec5\_id, ...],

'2600m': [rec4\_id, ...],

...

}

"""

block\_dict = {} # The dictionary with blocks to be generated and returned

print('Run simple blocking:')

print(' List of blocking key attributes: '+str(blk\_attr\_list))

print(' Number of records to be blocked: '+str(len(rec\_dict)))

print('')

for (rec\_id, rec\_values) in rec\_dict.items():

rec\_bkv = '' # Initialise the blocking key value for this record

# Process selected blocking attributes

#

for attr in blk\_attr\_list:

attr\_val = rec\_values[attr]

rec\_bkv += attr\_val

# Insert the blocking key value and record into blocking dictionary

#

if (rec\_bkv in block\_dict): # Block key value in block index

# Only need to add the record

#

rec\_id\_list = block\_dict[rec\_bkv]

rec\_id\_list.append(rec\_id)

else: # Block key value not in block index

# Create a new block and add the record identifier

#

rec\_id\_list = [rec\_id]

block\_dict[rec\_bkv] = rec\_id\_list # Store the new block

return block\_dict

# -----------------------------------------------------------------------------

def phoneticBlocking(rec\_dict, blk\_attr\_list):

"""Build the blocking index data structure (dictionary) to store blocking

key values (BKV) as keys and the corresponding list of record identifiers.

A blocking is implemented that concatenates Soundex encoded values of

attribute values.

Parameter Description:

rec\_dict : Dictionary that holds the record identifiers as keys

and corresponding list of record values

blk\_attr\_list : List of blocking key attributes to use

This method returns a dictionary with blocking key values as its keys and

list of record identifiers as its values (one list for each block).

"""

block\_dict = {} # The dictionary with blocks to be generated and returned

print('Run phonetic blocking:')

print(' List of blocking key attributes: '+str(blk\_attr\_list))

print(' Number of records to be blocked: '+str(len(rec\_dict)))

print('')

for (rec\_id, rec\_values) in rec\_dict.items():

rec\_bkv = '' # Initialise the blocking key value for this record

# Process selected blocking attributes

#

for attr in blk\_attr\_list:

attr\_val = rec\_values[attr]

if (attr\_val == ''):

rec\_bkv += 'z000' # Often used as Soundex code for empty values

else: # Convert the value into its Soundex code

attr\_val = attr\_val.lower()

sndx\_val = attr\_val[0] # Keep first letter

for c in attr\_val[1:]: # Loop over all other letters

if (c in 'aehiouwy'): # Not inlcuded into Soundex code

pass

elif (c in 'bfpv'):

if (sndx\_val[-1] != '1'): # Don't add duplicates of digits

sndx\_val += '1'

elif (c in 'cgjkqsxz'):

if (sndx\_val[-1] != '2'): # Don't add duplicates of digits

sndx\_val += '2'

elif (c in 'dt'):

if (sndx\_val[-1] != '3'): # Don't add duplicates of digits

sndx\_val += '3'

elif (c in 'l'):

if (sndx\_val[-1] != '4'): # Don't add duplicates of digits

sndx\_val += '4'

elif (c in 'mn'):

if (sndx\_val[-1] != '5'): # Don't add duplicates of digits

sndx\_val += '5'

elif (c in 'r'):

if (sndx\_val[-1] != '6'): # Don't add duplicates of digits

sndx\_val += '6'

if (len(sndx\_val) < 4):

sndx\_val += '000' # Ensure enough digits

sndx\_val = sndx\_val[:4] # Maximum length is 4

rec\_bkv += sndx\_val

# Insert the blocking key value and record into blocking dictionary

#

if (rec\_bkv in block\_dict): # Block key value in block index

# Only need to add the record

#

rec\_id\_list = block\_dict[rec\_bkv]

rec\_id\_list.append(rec\_id)

else: # Block key value not in block index

# Create a new block and add the record identifier

#

rec\_id\_list = [rec\_id]

block\_dict[rec\_bkv] = rec\_id\_list # Store the new block

return block\_dict

# -----------------------------------------------------------------------------

def slkBlocking(rec\_dict, fam\_name\_attr\_ind, giv\_name\_attr\_ind,

dob\_attr\_ind, gender\_attr\_ind):

"""Build the blocking index data structure (dictionary) to store blocking

key values (BKV) as keys and the corresponding list of record identifiers.

This function should implement the statistical linkage key (SLK-581)

blocking approach as used in real-world linkage applications:

http://www.aihw.gov.au/WorkArea/DownloadAsset.aspx?id=60129551915

A SLK-581 blocking key is the based on the concatenation of:

- 3 letters of family name

- 2 letters of given name

- Date of birth

- Sex

Parameter Description:

rec\_dict : Dictionary that holds the record identifiers as

keys and corresponding list of record values

fam\_name\_attr\_ind : The number (index) of the attribute that contains

family name (last name)

giv\_name\_attr\_ind : The number (index) of the attribute that contains

given name (first name)

dob\_attr\_ind : The number (index) of the attribute that contains

date of birth

gender\_attr\_ind : The number (index) of the attribute that contains

gender (sex)

This method returns a dictionary with blocking key values as its keys and

list of record identifiers as its values (one list for each block).

"""

block\_dict = {} # The dictionary with blocks to be generated and returned

print('Run SLK-581 blocking:')

print(' Number of records to be blocked: '+str(len(rec\_dict)))

print('')

for (rec\_id, rec\_values) in rec\_dict.items():

rec\_bkv = '' # Initialise the blocking key value for this record

# Get family name value

#

fam\_name = rec\_values[fam\_name\_attr\_ind]

if (fam\_name == ''):

rec\_bkv += '999'

else:

fam\_nam = fam\_name.replace('-','') # Remove non letter characters

fam\_nam = fam\_name.replace(",",'')

fam\_nam = fam\_name.replace('\_','')

if (len(fam\_name) >= 5):

rec\_bkv += (fam\_name[1]+fam\_name[2]+fam\_name[4])

elif (len(fam\_name) >= 3):

rec\_bkv += (fam\_name[1]+fam\_name[2]+'2')

elif (len(fam\_name) >= 2):

rec\_bkv += (fam\_name[1]+'22')

# Get given name value

#

giv\_name = rec\_values[giv\_name\_attr\_ind]

if (giv\_name == ''):

rec\_bkv += '99'

else:

giv\_nam = giv\_name.replace('-','') # Remove non letter characters

giv\_nam = giv\_name.replace(",",'')

giv\_nam = giv\_name.replace('\_','')

if (len(giv\_name) >= 3):

rec\_bkv += (giv\_name[1]+giv\_name[2])

elif (len(giv\_name) >= 2):

rec\_bkv += (giv\_name[1]+'2')

# DoB structure we use: dd/mm/yyyy

# Get date of birth

#

dob = rec\_values[dob\_attr\_ind]

dob\_list = rec\_values[dob\_attr\_ind].split('/')

# Add some checks

#

if (len(dob\_list[0]) < 2):

dob\_list[0] = '0' + dob\_list[0] # Add leading zero for days < 10

if (len(dob\_list[1]) < 2):

dob\_list[1] = '0' + dob\_list[1] # Add leading zero for months < 10

dob = ''.join(dob\_list) # Create: ddmmyyyy

assert len(dob) == 8, dob

rec\_bkv += dob

# Get gender

#

gender = rec\_values[gender\_attr\_ind].lower()

if (gender == 'm'):

rec\_bkv += '1'

elif (gender == 'f'):

rec\_bkv += '2'

else:

rec\_bkv += '9'

# Insert the blocking key value and record into blocking dictionary

#

if (rec\_bkv in block\_dict): # Block key value in block index

# Only need to add the record

#

rec\_id\_list = block\_dict[rec\_bkv]

rec\_id\_list.append(rec\_id)

else: # Block key value not in block index

# Create a new block and add the record identifier

#

rec\_id\_list = [rec\_id]

block\_dict[rec\_bkv] = rec\_id\_list # Store the new block

return block\_dict

# -----------------------------------------------------------------------------

# Extra task: Implement canopy clustering based blocking as described in

# the Data Matching book

# -----------------------------------------------------------------------------

def printBlockStatistics(blockA\_dict, blockB\_dict):

"""Calculate and print some basic statistics about the generated blocks

"""

print('Statistics of the generated blocks:')

numA\_blocks = len(blockA\_dict)

numB\_blocks = len(blockB\_dict)

block\_sizeA\_list = []

for rec\_id\_list in blockA\_dict.values(): # Loop over all blocks

block\_sizeA\_list.append(len(rec\_id\_list))

block\_sizeB\_list = []

for rec\_id\_list in blockB\_dict.values(): # Loop over all blocks

block\_sizeB\_list.append(len(rec\_id\_list))

print('Dataset A number of blocks generated: %d' % (numA\_blocks))

print(' Minimum block size: %d' % (min(block\_sizeA\_list)))

print(' Average block size: %.2f' % \

(float(sum(block\_sizeA\_list)) / len(block\_sizeA\_list)))

print(' Maximum block size: %d' % (max(block\_sizeA\_list)))

print('')

print('Dataset B number of blocks generated: %d' % (numB\_blocks))

print(' Minimum block size: %d' % (min(block\_sizeB\_list)))

print(' Average block size: %.2f' % \

(float(sum(block\_sizeB\_list)) / len(block\_sizeB\_list)))

print(' Maximum block size: %d' % (max(block\_sizeB\_list)))

print('')

# -----------------------------------------------------------------------------

# End of program.