BioBank Program Documentation

Basic BioBank Extract
The program can be modified by directly writing code into the the following functions, without changing the arguments and returned variables:

def process_file(headers, output_column_names):
"""This function is used to find and specify which fields in Biobank you are interested in and what columns you want in your output file. You may modify this function."""

Args:
- headers (These are all of the available fields in the main biobank file)
- output_column_names (A dictionary that is initialized with Patient_ID. Add the column names for your output to this list.)

Returns:
- field_index (dict required - Assign the a key name to the fields associated with that name)
- output_column_names (list required - expects 'Patient_ID')

Examples:
- field_index['ICD10'] = opf.get_header_indices(headers, '41202') + opf.get_header_indices(headers, '41204')
  - On the Biobank Site, you will see these field numbers associated with main and secondary ICD10 diagnoses. You can now refer to field_index['ICD10'] for all icd10 codes associated with '41202' and '41204'.
- output_column_names.append('Has_ICD10_Codes')
  - This adds the column name, 'Has_ICD10_Codes' to your output file.

def next_row(line, output_row, output_column_names, field_index):
"""This function lets you handle each row in the biobank file one at a time. Expects a dictionary of values that will become one line of output in your output file. All biobank values will be of type 'string', make sure to convert to float or int when comparing numerical values but not codes that may start with 0 (e.g. - ICD9). You may modify this function."""

Args:
- line (List of Values in the Biobank file for this particular patient)
- output_row (A dictionary that is already initialized with the

Patient_ID of this row)
- output_column_names (The column names you created from the above file)
- field_index (The indices you created from the above file)

Returns:
- output_row (dict required - values should be assigned for each column in your output_column_names)

Examples:
- icd10_values = main_func.get_patient_vals(headers, field_index['ICD10'])
  - Gets all values associated with this field for this particular patient
- output_row['Has_ICD10_Codes'] = 1
  - We are assigning a '1' to this patient for this column.

Basic HES Extract

The program can be modified by directly writing code into the the following functions, without changing the arguments and returned variables:

def process_file(headers):

"""This function is used to find and specify which fields in Biobank you are interested in and what columns you want in your output file. You may modify this function."""

Args:
- headers (These are all of the available fields in the HES file)

Returns:
- field_index (dict required - Assign the a key name to the fields associated with that name)
- groupby_dict (dict required - There can be multiple admissions per patient. This will group the values accordingly)

Examples:
- field_index['ICD10'] = opf.get_header_indices(headers, 'diag_icd10')
  - On the Biobank Site, you will see these field numbers associated with main and secondary ICD10 diagnoses. You can now refer to field_index['ICD10'] for all icd10 codes associated with '41202' and '41204'.
- groupby_dict['Has_ICD10_Codes'] = 'concat'
  - This will concatenate all of the codes the patient has with a comma like this: code1,code2,code3 at the end of the program.
  - The accepted kind grouping is: sum, mean, min, max, concat

def next_row(line, output_row, output_column_names, field_index):

"""This function lets you handle each row in the biobank file one at a time. Expects a dictionary of values that will become one line of output in your output file. All biobank values will be of type 'string', make sure to convert to float or int when comparing numerical values but not codes that may start with 0 (e.g. - ICD9). You may modify this function."""

Args:

- line (List of Values in the Biobank file for this particular patient)
- output_row (A dictionary that is already initialized with the Patient_ID of this row)
- output_column_names (The column names you created from the above file)
- field_index (The indices you created from the above file)

Returns:

- output_row (dict required - values should be assigned for each column in your output_column_names)

Examples:

- icd10_values = main_func.get_patient_vals(headers, field_index['ICD10'])
  - Gets all values associated with this field for this particular patient
- output_row['Has_ICD10_Codes'] = code1,code2,code3
  - We assign the codes the patient has for this row.

main_func.py

def get_header_indices(headers, bb_field, bb_field_age=None):

""" Returns a list of indices or a list of tuples with indices that correspond to specific fields in the file"""

Args:

- headers (These are the all of the available fields in biobank)
- bb_field (The code associated with this BioBank Field)
- bb_field_age (The code associated with the age of the above field - Not Required - Default:None)

Returns:

- field_indices (Returns list of code positions or list of tuples with positions as (bb_field, bb_field_age))

Examples:

- get_header_indices(headers, “41202”) + get_header_indices(headers, '41204')
  - ICD10 codes have field codes of 41202 and 41204
- get_header_indices(headers, “6560_0”)
  - If you only want values from the first hospital visit for some random field: 6560.
- get_header_indices(headers, '20002', '20009')
  - If you want self-reported medical codes and associated ages, OR any code with another associated code. (e.g. - age at 20009_0_0 was the age for diagnosis code at 20002_0_0.

def get_patient_vals(line, index_list, return_both = False):

"""Returns patient values that correspond to the given indices"""

Args:

- line (These are the all values for one row in the file)
- index_list (Excepts values from get_header_indices)
- return_both (Returns both the code and age value if the index_list is also a tuple - Not Required - Default:False)

Returns:

- List of Values corresponding to the index_list
-

Examples:

- get_patient_vals(line, field_index[‘ICD10’])
  - Gets the patient values associated with the field index you specified in the process_file() function.
- get_patient_vals(line, field_index[‘Self_Reported_Medical_Diag’], return_both=True)
  - If return_both is true, and your field_index is a list of tuples with the indices of field_one mapped to the indices at field_two, a list of tuples with the associated values will be returned. Otherwise, only the values for field_one will be returned.

def ends_with(values, partial_string, all_vals=True):

"""Returns a boolean (True/False) if values end with the partial string“””

Args:

- values (These are a list of values. You can input any kind of values as long as they're all strings)
- partial_string (The function will check for string values that end with this partial_string)
- all_vals (If true, will only return True if there is at least one value and all available values end with the partial string - Not Required - Default:True)

Returns:

- Boolean (True/False)

Examples:

- ends_with(patient_malformations, '_stroke')
  - If you've translated a patient's codes to pre-defined stroke malformations and they have '_stroke' at the end (e.g. - "ischemic_stroke, 'hemorrhagic_stroke'). This example will return True if the patient only has stroke malformations, and False if the patient has no malformations or at least one other kind of malformation.
- ends_with(patient_malformations, '_stroke', False)
  - In the same scenario, this example will return True if the patient has AT LEAST ONE stroke malformation.

def coexist(tuple_list, main_values, conditional_values, return_existing_cond = False):

“”” Say you have a list of tuples. Each tuple contains two values which may co-exist. This function returns the values that coexist based on the values the patient has””

Args:

- tuple_list (A list or set of tuples where the first and second value may co-exist)
- main_values (A list of patient values matching the first value in a tuple)
- conditional_values (A list of patient values matching the second value in a tuple)
- return_existing_cond (If true, returns both main and conditional values where paired tuple values exist. Otherwise, returns just main_values - Not Required - Default:False)

Returns:

- set of main_values where the condition is true OR

- (If return_existing_cond is True) set of main_values and set of conditional_values

Examples:

- coexist(cant_coexist_tuples, patient_main_malformations, patient_conditional_malformations)
  - Say you've predefined a set of malformations for CHD and a conditional set of malformations that a patient could have. This function will return the patient_main_malformations if the CHD malformation and it's conditional malformation co-exist.
- coexist(coexist_malformations, patient_malformations, patient_malformations, True)
  - You can use the same set of malformations for main and conditional values. This will return both the first and second set of malformations that match a tuple instance.

def match_codes(key_codes, patient_codes):

"""" The key_codes will be partially matched to the patient_codes and return the values of those keys if key_codes is a dictionary. Otherwise it will return the matching codes. A partial match means: "E50" can match patient values "E500", "E501", and "E50" for example """"

Args:

- key_codes (A list or a dict where the keys may match patient codes)
- patient_codes (A list of codes that the patient has)

Returns:

- matched_codes (either translated values or the codes themselves)

Examples:

- match_codes(icd10_codes, patient_icd10_codes)
  - This will return the ICD10 codes that the patient has.
- match_codes(icd10_malformations, patient_icd10_codes)
  - This will return the corresponding malformations to the ICD10 codes that a patient has.

def get_min_age(patient_code_age_tuples, key_codes, age_criteria_dict = None):

"""" Returns the Minimum age and codes where the codes match the key codes""

Args:

- patient_code_age_tuples (A list or set of tuples where the first

value is a patient's code and the second value is the age at which that patient was diagnosed with that code)
- key_codes (A list of patient values matching the first value in a tuple)
- age_criteria_dict (The function will only output codes if the age meets the criteria specified in this dictionary. Criteria is met if **age is LESS than age criteria**. Not all codes require an age criteria - Not Required - Default:None)

Returns:
- min_age (earliest age at which patient was diagnosed with any of these codes)
- available_codes (codes for which the patient was diagnosed (useful if age_criteria_dict exists))

Examples:

- get_min_age(code_age_tuples, self_reported_codes, age_criteria)
  - We will get the minimum age and code for all codes in self_reported_codes that exist for this patient. If the code has an age_criteria, we will only include the code if it is LESS than the age in age_criteria.

def single_output_conversion(list_of_values, conversion=None, default_value = None, input_value=None):

"""Converts a single value or a list of values into a different value for readability and analysis"""

Args:
- list_of_values (List or set of patient values)
- conversion (A dictionary where the values will be matched to the patient values and output is the key - Not Required - Default:None).
- default_value (A default value for conversions - Not Required - Default:None)
- input_value (Determines how the values are read in - Not Required - Default:None)
  - **A:** Converts after looking at ALL values
  - **F:** Converts after looking at the FIRST non-blank value

Returns:
- return_value (a single value that may or may not be converted)

Examples:

- single_output_conversion(patient_education_values, {'College': 1}, 'NoCollege', 'A')

- Looks at all values in the specified education fields and looks for a 1. If the patient has a 1, the output is College. Otherwise, it's 'NoCollege'.

- single_output_conversion(patient_bmi_values, 'F')
  - Will return the first available value for BMI. This reduces the number of blank BMI values and will output BMI upon admission (for most patients)
- single_output_conversion(patient_smoking_status_values, {'Never': 0, 'Previous': 1, 'Current': 2, 'NA':-3})
  - Will convert the numerical values found into their string values.

sheets.py

def to_Dict(sheet_dict):

""" Will convert values that appear to be dictionaries into Python dictionaries. All values inside the dictionary will be converted to lists with string values. """

Args:

- sheet_dict (A string representation of a dictionary. Useful if directly inputting dictionaries into Excel file).

Returns:

- sheet_dict (As dictionary - All keys and values are converted into type 'string')

def sheet_to_dict(reader, sheet, keys_column, value_column, converters = None, dict_values = False):

""" The program outputs a dictionary based on two columns in an Excel Sheet."""

Args:

- reader (Must be a pandas ExcelFile)
- sheet (Must be a sheet in the above file)
- keys_column (Must be a column In the sheet. The rows will become the keys in the output dictionary)
- value_column (Must be a column in the sheet. The rows will become the values for each key in the output dictionary)
- converters (Some values may automatically be read in as type int or string. Make sure to specify to prevent errors later in the program. For example: ICD9 codes can start with zeroes (01129). Converting ICD9 codes to string prevents the zeroes from

disappearing - Not Required - Default:None)
- dict_values (If true, will expect a string representation of a dictionary in the row values and convert it into an actual dictionary).

Returns:
- sheet_dict (keys are mapped to column values row-wise in the Excel sheet)

Examples:

- sheet_to_dict(excel_reader, ‘Attributes_Sheet’, 'Attribute', ‘Conversion_Dict’, dict_values=True)
  - Will return a dictionary where the values in conversion_dict will be returned as actual dictionaries, rather than string representations of them
- sheet_to_dict(excel_reader, ‘ICD9_Malformations’, 'ICD9', 'MALFORMATION', {'ICD9': str})
  - Will return a dictionary where the codes in ICD9 column are forcefully converted into type string, rather than type int

def sheet_to_tuple(reader, sheet, keys_column, value_column, converters = None):

""" The program outputs a list of tuples based on two columns in an Excel Sheet.”””

Args:
- reader (Must be a pandas ExcelFile)
- sheet (Must be a sheet in the above file)
- keys_column (Must be a column In the sheet. The rows will become the first value in a tuple)
- value_column (Must be a column in the sheet. The rows will become the second value in a tuple)
- converters (Some values may automatically be read in as type int or string. Make sure to specify to prevent errors later in the program. For example: ICD9 codes can start with zeroes (01129). Converting ICD9 codes to string prevents the zeroes from disappearing - Not Required - Default:None)

Returns:
- sheet_tuple (keys and values are coupled together row-wise in the Excel sheet)

QualityCheck.py

Useful to compare the counts of two separate extracts. Should be used after every extract before analysis.