```
# Author: J. Connolly: updated 5/30/2017
# ##############General notes
# All hard coded columns are purposefully, these are new columns used for tracking and grouping
#
# When collapsing bin assignments, all current assignment are considered and used to generated a single species
# assignment. You will lose huc specific assignments. Typically this is only done for new huc 2 where we do not have an
# assignment. Relies on the assumption that higher coded values should trump lower coded values.
#     collapse_huc = True
#     collapse_new_HUCS_only = True
#
# If you opt to collapse across all HUCs for a species you will violate the assumption that higher values trump lower
# values. As a result you will **need to updated the DB_code_Dict in Step_3_ReCode Bin Table** so that the values
# starting with 13 are 'Yes' and not 'No'. These are huc specific assignment for land locked hucs, species can only be
# in these bins in coastal HUC2s.

#  ############## ASSUMPTIONS
# Final columns headers related to bin information are the same as those in current_bin_table
# Bin input table columns are in the correct order ie, species info cols, bins, database info columns
# HUC-2 15 should be considered land-locked because it is land-locked within US jurisdiction
# ***Coded bins values are hierarchical, a higher number trumps a lower number when collapsing***
#       ## NOTE if collapsing across all HUCs for a species DB_code_Dict will need to be updated in
#         Step_3_ReCode Bin Table

import pandas as pd
import os
import datetime
import sys

# #############  User input variables
# location where out table will be saved
table_folder = r'C:\Users\JConno02\Documents\Projects\ESA\Bins\updates\Script_Check_20170627'
# Boolean variables to generates a bin assignment for the species based on all current assignments across HUCS
# If only collapse_huc is True HUC2 specific bin assignment will be lost for the whole species
# If both are true collapsed species bin assignment would only be applied to an added HUC2
# Both False bin assignments for new hucs will be blank
collapse_huc = True
collapse_new_HUCS_only = True  # if true collapse_huc must be true

# Active species master list with supporting information
# Note extinct species will be included in the output table
# Place master species list into the archive folder for tracking found at table_folder
current_master_nm = 'MasterListESA_Feb2017_20170410.csv'
entity_id_col_m = 'EntityID'
```

```
44 lead_agency_col = 'Lead Agency'
45 # WoE group crosswalk used in post-processor
46 # Place WoE group crosswalk into the archive folder for tracking found at table_folder
47 woe_csv_nm = r'species woe taxa.csv'
48 entity_id_col_woe = 'EntityID'
49 esa_woe_group_col = 'ESA Group'
50
51 # Bin table actively being use - INPUT SOURCE - Bin Database
52 # Place current bin into the archive folder for tracking found at table_folder
53 current_bin_table_name = 'Current_Assignments_20170420.csv'
54 entity_id_col_c = 'EntityID'  # column header species EntityID current_bin_table
55 huc2_col_c = 'HUC_2'  # column header species HUC_2 current_bin_table
56 # column headers bins current_bin_table
57 bin_col = ['Terrestrial Bin', 'Bin 1', 'Bin 2', 'Bin 3', 'Bin 4', 'Bin 5', 'Bin 6', 'Bin 7', 'Bin 8', 'Bin 9', 'Bin 10']
58
59 # HUC2s removed from species range based on updated GIS files
60 # INPUT SOURCE - output table 'removed_composite.csv' from HUC2 assignment script
61 # Place removed_huc_table into the archive folder for tracking found at table_folder
62 removed_huc_table_name = 'removed_Fall2016.csv'
63 entity_id_col_r = 'Entity_id'  # column header species EntityID removed_huc_table
64 huc2_col_r = 'Huc_2'  # column header species HUC_2 removed_huc_table
65
66 # HUC2s added from species range based on updated GIS files
67 # INPUT SOURCE - output table 'added_composite.csv' from HUC2 assignment script
68 # Place removed_huc_table into the archive folder for tracking found at table_folder
69 add_huc_table_name = 'added_Fall2016.csv'
70 entity_id_col_a = 'Entity_id'  # column header species EntityID add_huc_table
71 huc2_col_a = 'Huc_2'  # column header species HUC_2 add_huc_table
72
73 # The species entityID is the tracking number by population across all ESA tools.  This number is pulled from
   the FWS
74 # TESS database. NMFS species not entered in the TESS database are given a place holder tracker that started
   with NMFS.
75 # Updates to this identifier are rare but **MUST** be taken into account at the **START** of all updates in
   order to
76 # make sure all files and species information are linked correctly.- INPUT SOURCE- User
77 entityid_updated = {'NMFS88': '9432', 'NMFS180': '11353', 'NMFS181': '11355', '5623': '11356', 'NMFS22': '10377'}
78
79 # Dictionary of updates to bin tracking codes
80 # If bin code starts with 13, this is represents a marine bin where the species occurs for coastal hucs. These
   bins do
81 # do not occur in the land-locked HUC2s
82 # ##INPUT SOURCE - User -keys are string, values are integers
83 bin_code_update = {'7': 137, '12': 1312, '2': 132, '6': 136, '28': 1328, '29': 1329, '210': 13210, '211': 13211}
84
85 # ############ Static input variables
86 today = datetime.datetime.today()
87 date = today.strftime('%Y%m%d')
88
```

```
89  land_locked_hucs = ['4', '5', '6', '7', '9', '10', '11', '14', '15', '16']  # HU2-15 is land-lock within the US
90  marine_bins = ['8', '9', '10']  # marine bins not directly modelled
91  archived_location = table_folder + os.sep + 'Archived'  # scratch workspace
92  os.mkdir(archived_location) if not os.path.exists(archived_location) else None
93
94  current_bin_table = archived_location + os.sep + current_bin_table_name
95  removed_huc_table = archived_location + os.sep + removed_huc_table_name
96  add_huc_table = archived_location + os.sep + add_huc_table_name
97  current_master = archived_location + os.sep + current_master_nm
98  woe_csv = archived_location + os.sep + woe_csv_nm
99  archived_file = archived_location + os.sep + "DroppedSpecies_" + str(date) + '.csv'  # Species removed from
    master list
100 no_bins_file = archived_location + os.sep + "Species_w_NewHUC2_" + str(date) + '.csv'  # Species/HUC w/o
    bin assignment
101 removed_hucs = archived_location + os.sep + "RemovedHUC2_" + str(date) + '.csv'  # HUC2 no longer found in
    species range
102 wide_woe = archived_location + os.sep + "WideWoeGroups_" + str(date) + '.csv'  # Wide format of WoE group
    crosswalk
103 outfile = archived_location + os.sep + "UpdatedBins_" + str(date) + '.csv'  # working bin tables with updates
104
105
106 def load_data(current, removed, added, master_sp, woe_table):
107     # VARS: current: current bin table in use; removed: list of huc 2 being removed by species; added: list of
    huc2
108     # being added by species; master_sp current species master list; woe_table: woe group crosswalk from
    post-processor
109     # DESCRIPTION: removes columns without headers from all data frames; sets entity id col as str in all tables;
110     # updates entity ids as needed; generated list of col headers all data frames; EntityID col head standardize
    to
111     # match the master species data frame and HUC2 col header standardized to match the active bin data
    frame;
112     # ['Spe_HUC'] (entityid_huc) added to all tables as a unique identifier common across tables. Try/Excepts
    makes sure
113     # we have a complete archive of data used for update, and intermediate tables.
114     # RETURN: data frames of inputs tables; KEY col headers standardize; entity ids updated when needed
115
116     try:
117         c_df = pd.read_csv(current)
118     except IOError:
119         print('\nYou must move the current bin table to Archived folder for this update')
120         sys.exit()
121
122     [c_df.drop(m, axis=1, inplace=True) for m in c_df.columns.values.tolist() if m.startswith('Unnamed')]
123     c_df[str(entity_id_col_c)] = c_df[str(entity_id_col_c)].astype(str)
124     c_cols = c_df.columns.values.tolist()
125     c_cols = update_columns_header(str(entity_id_col_c), str(entity_id_col_m), c_cols)
126     c_df.columns = c_cols
127     c_cols.append('Updated') if 'Updated' not in c_cols else None
128     c_df = c_df.reindex(columns=c_cols)
```

```
129
130     wide_cols = c_df.columns.values.tolist()
131     bin_loc_start = wide_cols.index((bin_col[0]))
132     # use to index list of bin values; add 1 because when sub-setting output does not include value in the last
    index
133     # position but everything in front of it
134     bin_loc_end = (wide_cols.index((bin_col[len(bin_col) - 1]))) + 1
135     [c_df[str(entity_id_col_c)].replace(z, entityid_updated[z], inplace=True) for z in entityid_updated.keys()]
136     c_df['Spe_HUC'] = c_df[str(entity_id_col_c)].astype(str) + "_" + c_df[str(huc2_col_c)].astype(str)
137
138     try:
139         remove_df = pd.read_csv(removed)
140     except IOError:
141         print('\nYou must move the removed huc table to Archived folder for this update')
142         sys.exit()
143
144     [remove_df.drop(z, axis=1, inplace=True) for z in remove_df.columns.values.tolist() if z.startswith('Unnamed'
    )]
145     remove_df[str(entity_id_col_r)] = remove_df[str(entity_id_col_r)].astype(str)
146     [remove_df[str(entity_id_col_r)].replace(z, entityid_updated[z], inplace=True) for z in entityid_updated.keys
    ()]
147     remove_df['Spe_HUC'] = remove_df[str(entity_id_col_r)].astype(str) + "_" + remove_df[str(huc2_col_r)].
    astype(str)
148     removed_cols = remove_df.columns.values.tolist()
149     removed_cols = update_columns_header(str(entity_id_col_r), str(entity_id_col_m), removed_cols)
150     removed_cols = update_columns_header(str(huc2_col_r), str(huc2_col_c), removed_cols)
151     remove_df.columns = removed_cols
152
153     try:
154         add_df = pd.read_csv(added)
155     except IOError:
156         print('\nYou must move the added huc table to Archived folder for this update')
157         sys.exit()
158
159     [add_df.drop(a, axis=1, inplace=True) for a in add_df.columns.values.tolist() if a.startswith('Unnamed')]
160     add_df[str(entity_id_col_a)] = add_df[str(entity_id_col_a)].astype(str)
161     [add_df[str(entity_id_col_a)].replace(z, entityid_updated[z], inplace=True) for z in entityid_updated.keys()]
162     add_df['Spe_HUC'] = add_df[str(entity_id_col_a)].astype(str) + "_" + add_df[str(huc2_col_a)].astype(str)
163     add_cols = add_df.columns.values.tolist()
164     add_cols = update_columns_header(str(entity_id_col_a), str(entity_id_col_m), add_cols)
165     add_cols = update_columns_header(str(huc2_col_a), str(huc2_col_c), add_cols)
166     add_df.columns = add_cols
167
168     try:
169         sp_df = pd.read_csv(master_sp)
170     except IOError:
171         print('\nYou must move the master species table to Archived folder for this update')
172         sys.exit()
173     try:
```

```
174         woe_df = pd.read_csv(woe_table)
175         woe_df[str(entity_id_col_woe)] = woe_df[str(entity_id_col_woe)].astype(str)
176         woe_cols = woe_df.columns.values.tolist()
177         woe_cols = update_columns_header(str(entity_id_col_woe), str(entity_id_col_m), woe_cols)
178         woe_df.columns = woe_cols
179     except IOError:
180         print('\nYou must move the woe group crosswalk table to Archived folder for this update')
181         sys.exit()
182 
183     return c_df, remove_df, add_df, sp_df, bin_loc_start, bin_loc_end, woe_df
184 
185 
186 def set_common_cols(bin_df, sp_df):
187     # VARS: current:master_entid: Species identifier master species list; bin_entid: Species identifier active bin
    table
188     # bin_df: data frame of active bin table; sp_df: data frame of master species list
189     # DESCRIPTION: Checks with user to see which columns from the master species list should be included on
    the updated
190     # bin table
191     # RETURN: list of columns to be included
192 
193     sp_df_cols = sp_df.columns.values.tolist()
194     bin_df_cols = bin_df.columns.values.tolist()
195     common_cols = [j for j in sp_df_cols if j in bin_df_cols]
196 
197     poss_answer = ['Yes', 'No']
198     ask_q = True
199     while ask_q:
200         user_input = raw_input(
201             'Are these are the columns that should be updated from master species table? {0}: Yes or No: '.format
    (
202                 common_cols))
203         if user_input not in poss_answer:
204             print 'This is not a valid answer: type Yes or No'
205         elif user_input == 'Yes':
206             break
207         else:
208             additional_cols = raw_input('Which additional columns should be included - {0}: '.format(sp_df_cols))
209             if type(additional_cols) is str:
210                 additional_cols = additional_cols.split(",")
211                 additional_cols = [j.replace("' ", "") for j in additional_cols]
212                 additional_cols = [j.replace("'", "") for j in additional_cols]
213                 additional_cols = [j.lstrip() for j in additional_cols]
214                 common_cols.extend(additional_cols)
215     return common_cols
216 
217 
218 def update_columns_header(current_col, updated_col, list_of_col):
219     # VARS: current:current_col: value in current list, updated_col: value it should be Fd to, list_of_col: list of
```

```
220     # values
 221    # DESCRIPTION: Standardize column headers terminology across tables based on user input variables.
 222    # RETURN: Update list of values to be used as column headers for data frames
 223
 224    if current_col != updated_col:
 225        loc = list_of_col.index(str(current_col))
 226        list_of_col.remove(str(current_col))
 227        list_of_col.insert(loc, str(updated_col))
 228        return list_of_col
 229    else:
 230        return list_of_col
 231
 232
 233 def add_groupby_columns(row, df):
 234    # VARS: row: row: identifier used for grouping (species entityid, df: working df with bin codes
 235    # DESCRIPTION:  For each entityid in WoE df, filters the WoE df to include all occurrences of the entityID,
 236    # flags row as WoE_group_1, _2 or _3 depending on the number of WoE groups the species occurs in. This
        sets up the
 237    # data frame allowing a pivot to convert format to wide format ie each entityID is a single row with multiple
 238    # columns if the species is in multiple groups. (Input table is long format, each row is a unique species WoE
        group
 239    # combo but the same entity ID can be found on multiple rows.)
 240    # RETURN: None groups applied directly in df
 241
 242    filter_df = df.loc[df[entity_id_col_m] == row]
 243
 244    if len(filter_df) == 2:
 245        df.loc[df[entity_id_col_m] == row, 'Woe_Group'] = ['WoE_group_1', 'WoE_group_2']
 246    elif len(filter_df) == 1:
 247        df.loc[df[entity_id_col_m] == row, 'Woe_Group'] = ['WoE_group_1']
 248    elif len(filter_df) == 3:
 249        df.loc[df[entity_id_col_m] == row, 'Woe_Group'] = ['WoE_group_1', 'WoE_group_2', 'WoE_group_3']
 250
 251
 252 def check_final_col_order(cur_cols, updated_cols):
 253    # VARS: current:cur_col: order of cols in current wide bin table, updated_cols: columns identified by use to
        include
 254    # in output
 255    # DESCRIPTION: Set the desired order for the columns in the output table
 256    # RETURN: columns headers to be used as reindex for the output table
 257
 258    for r in updated_cols:
 259        if r not in cur_cols:
 260            index_location = raw_input(
 261                'What is the index position where this column should be inserted- based 0 {0}: '.format(r))
 262            cur_cols.insert(int(index_location), r)
 263
 264    poss_answer = ['Yes', 'No']
 265    ask_q = True
```

```
266     while ask_q:
267         user_input = raw_input('Is this the order you would like the columns to be {0}: Yes or No: '.format(
        cur_cols))
268         if user_input not in poss_answer:
269             print 'This is not a valid answer'
270         elif user_input == 'Yes':
271 
272             break
273         else:
274             cur_cols = raw_input('Please enter the order of columns comma sep str ')
275     if type(cur_cols) is str:
276         cur_cols = cur_cols.split(",")
277         cur_cols = [j.replace("' ", "") for j in cur_cols]
278         cur_cols = [j.replace("'", "") for j in cur_cols]
279         cur_cols = [j.lstrip() for j in cur_cols]
280     return cur_cols
281 
282 
283 def updates_hucs(working_df, added_df, removed_df, spe_df):
284     # VARS: working_df: df that is being updated; added_df: df of huc 2 being added by species, removed_df: df
        of huc 2
285     # being removed by species, spe_df: data frame will all current species info
286     # DESCRIPTION: Filters working df to just species with a new huc2 using the common column ['Spe_HUC']
        new hucs will
287     # not have a bin assignment. Saves table of just the new hucs with blank bin assignments; Filters out
        species/huc
288     # combos using ['Spe_HUC'] that are no longer in species range; Saves archive of removed hucs
289     # RETURN: working bin data frame
290 
291     all_columns = working_df.columns.values.tolist()
292     missing_from_bin_table = added_df.loc[~added_df['Spe_HUC'].isin(working_df['Spe_HUC'])]
293 
294     missing_df = missing_from_bin_table[[str(entity_id_col_c), str(huc2_col_c), 'Spe_HUC']]
295     missing_df = missing_df.reindex(columns=all_columns)
296     concat_df = pd.concat([working_df, missing_df], axis=0)
297 
298     updated_df = pd.DataFrame(concat_df)  # makes a copy of df with all values included based on defined
        indexing
299     missing_df = pd.merge(missing_df, spe_df, on=str(entity_id_col_m), how='outer')
300     missing_df.to_csv(no_bins_file)
301 
302     removed_entries = working_df.loc[working_df['Spe_HUC'].isin(removed_df['Spe_HUC'])]
303     removed_entries.to_csv(removed_hucs)
304     updated_df = updated_df.loc[~updated_df['Spe_HUC'].isin(removed_df['Spe_HUC'])]
305     return updated_df
306 
307 
308 def collapse_species(working_df, added_df, start_index, end_index):
309     # VARS: working_df: df that is being updated; added_df: df of huc 2 being added by species
```

```
310     # DESCRIPTION: Generates species bin assignment based on previous assignments across hucs. This
    species assignment
311     # is applied either to just the new HUCs that do not have a bin assignment or replaces all bin assignments
    across
312     # huc2s for the species based on the user inputs collapse_huc and collapse_new_HUCS_only. The single bin
    assignment
313     # is generated by loading the assignment for the first huc2 into a list based on the index positions of the bin
314     # columns, then comparing the values for each of the following huc2s and retaining max value seen for
    each bin as
315     # the master.  These single collapsed bin assignment for the species is applied to specific huc2s based on
    the user
316     # input for boolean variable [collapse_huc] and [collapse_new_HUCS_only.]
317     # RETURN: working bin data frame
318
319     ent_list = list(set(working_df[entity_id_col_c].astype(str).values.tolist()))
320     if collapse_new_HUCS_only:
321         list_added_species = list(set(added_df[entity_id_col_c].astype(str).values.tolist()))
322     else:
323         list_added_species = ent_list
324
325     for k in ent_list:
326
327         if k not in list_added_species:
328             pass
329         else:
330             ent = str(k)
331             lookup_huc_bins = working_df.loc[working_df[str(entity_id_col_c)] == ent]
332             spe_huc = lookup_huc_bins['Spe_HUC'].values.tolist()
333             list_spe_huc = lookup_huc_bins.values.tolist()
334             count_huc = len(list_spe_huc)
335             try:
336                 starting_values = map(int, list_spe_huc[0][int(start_index):int(end_index)])
337             except ValueError:  # this catches HUCs with blank assignments can't convert float nan to int
338                 starting_values = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
339
340             counter = 1  # counter set to 1 because index position 0 is used as the starting values
341             huc_specific_assignments = False
342
343             while counter < count_huc:
344                 try:
345                     current_bins = map(int, list_spe_huc[counter][int(start_index):int(end_index)])
346                 except ValueError:  # this catches HUCs with blank assignments can't convert float nan to int
347                     current_bins = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
348
349                 for x in current_bins:
350                     index_pos = current_bins.index(x)
351                     out_value = starting_values[index_pos]
352                     if out_value == x:
353                         pass
```

```
354                 elif out_value != x:
355                     update_value = max(out_value, x)  # coded bin values are hierarchical retain max val
356                     starting_values[index_pos] = update_value
357                     huc_specific_assignments = True
358
359             counter += 1
360
361         if huc_specific_assignments and collapse_new_HUCS_only:
362             for t in spe_huc:
363                 if t in added_df['Spe_HUC'].values.tolist():
364                     working_df.loc[working_df['Spe_HUC'] == t, ['Updated']] = 'New HUC add - Collapsed, ' \
365                                                               'huc specific bin assignments'
366                     working_df.loc[working_df['Spe_HUC'] == t, bin_col] = starting_values
367                 else:
368                     pass
369         elif huc_specific_assignments and not collapse_new_HUCS_only:
370             working_df.loc[working_df['Spe_HUC'].isin(spe_huc), ['Updated']] = 'All hucs collapsed, included huc ' \
371                                                               'specific bin assignments'
372             working_df.loc[working_df['Spe_HUC'].isin(spe_huc), bin_col] = starting_values
373
374         elif starting_values == [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]:
375             for t in spe_huc:
376                 if t in added_df['Spe_HUC'].values.tolist():
377                     working_df.loc[working_df['Spe_HUC'] == t, ['Updated']] = 'New Species/No previous assignment'
378
379                     working_df.loc[working_df['Spe_HUC'] == t, bin_col] = ['', '', '', '', '', '', '', '', '', '',
380                                                               '']
381                 else:
382                     pass
383         elif not huc_specific_assignments and collapse_new_HUCS_only:
384             for t in spe_huc:
385                 if t in added_df['Spe_HUC'].values.tolist():
386                     working_df.loc[working_df['Spe_HUC'] == t, ['Updated']] = 'New HUC add - Collapsed, No ' \
387                                                               'huc specific bin assignments'
388                     working_df.loc[working_df['Spe_HUC'] == t, bin_col] = starting_values
389                 else:
390                     pass
391
392         elif not huc_specific_assignments and not collapse_new_HUCS_only:
393             working_df.loc[working_df['Spe_HUC'].isin(spe_huc), ['Updated']] = 'All hucs collapsed, no huc ' \
394                                                               'specific bin assignments'
395
396         else:
397
398             working_df.loc[working_df['Spe_HUC'].isin(spe_huc), ['Updated']] = 'All hucs collapsed, no huc ' \
399                                                               'specific bin assignments'
400
```

```
401     return working_df
402
403
404 def check_land_locked_hucs(working_df):
405     # VARS: working_df: df that is being updated
406     # DESCRIPTION: Check the marine bin assignments for species found in the land locked hucs to make sure
    they are
407     # these huc are set to the values in the bin_code_update
408     # RETURN: working bin data frame
409
410     code_update = bin_code_update.keys()
411     for p in marine_bins:
412         column = 'Bin ' + p
413         for k in code_update:
414             update_value = bin_code_update[k]
415             working_df.loc[
416                 working_df[str(huc2_col_c)].isin(land_locked_hucs) & (working_df[str(column)] == int(k)), column]
    = int(
417                 update_value)
418     return working_df
419
420
421 # Time tracker
422 start_time = datetime.datetime.now()
423 print "Start Time: " + start_time.ctime()
424
425 # Step 1: Load data from current bin tables, tables used to update hucs and species info. Sets the columns
    from the
426 # species tables that should be included in the output bin tables.
427 df_current, df_remove, df_add, df_sp, bin_start_index, bin_end_index, df_woe = load_data(current_bin_table,
428                                                                                          removed_huc_table,
429                                                                                          add_huc_table, current_master,
430                                                                                          woe_csv)
431 cols_to_update = set_common_cols(df_current, df_sp)
432 df_spe_info = df_sp[cols_to_update]
433
434 # Step 2: Make species data frames from woe group crosswalk from post processor, converts the woe group
    crosswalk from
435 # long format to wide by adding a grouping category to the new [WoE_group] column for each entity id.
    Appends woe group
436 # in wide format to species info df. Saves a wide version of the woe groups to archive folder.
437 # WideWoeGroups_[date].csv - WoE group crosswalk in ide format
438 woe_col = df_woe.columns.values.tolist()
439 df_woe = df_woe.reindex(columns=woe_col.append('Woe_Group'))
440 [add_groupby_columns(species, df_woe) for species in df_woe[entity_id_col_m].values.tolist()]
441 pivot = (df_woe.pivot(index=entity_id_col_m, columns='Woe_Group', values=esa_woe_group_col)).reset_index(
    )
442 pivot.to_csv(wide_woe)
443
```

```
444 # Left outer join produces a complete set of records from Table A, with the matching records (where
    available) in Table
445 # B. If there is no match, the right side will contain null."
446 df_spe_info = pd.merge(df_spe_info, pivot, on=entity_id_col_m, how='left')
447 cols_to_update.extend(['WoE_group_1', 'WoE_group_2', 'WoE_group_3'])
448
449 # Step 3: Adds new HUCs and removes dropped HUCs based on species range update. Make archived of
    intermediate tables
450 # Species_w_NewHUC2_[date].csv - all blank bin assignments; new species and species with new HUCs
451 # RemovedHUC2_[date].csv - bins assignment for huc 2 dropped from species range
452
453 df_updated = updates_hucs(df_current, df_add, df_remove, df_spe_info)
454 df_land = pd.DataFrame(df_updated)  # make a copy of the df
455
456 # Step 4: Updated bins assignments in land locked hucs based on bin_code_update dictionary
457 df_final = check_land_locked_hucs(df_land)
458 df_final = pd.DataFrame(df_final)  # makes a copy of the df
459
460 # Step 5: Collapse bins assignment for a species across all hucs into a single assignment for a species,
    applies bin
461 # assignment to specific hucs based on the user input variables collapse_huc, collapse_new_HUCS_only
462
463 if collapse_huc:
464     df_collapse = pd.DataFrame(df_final)
465     del df_final
466     df_final = collapse_species(df_collapse, df_add, bin_start_index, bin_end_index)
467
468 # Step 6: Runs a final check on entity ids, removes old species info from bin data frame then merges the new
    species
469 # info to the bin data frame.
470
471 updated_entity_ids = entityid_updated.keys()
472 [df_final[str(entity_id_col_c)].replace(i, entityid_updated[i], inplace=True) for i in updated_entity_ids]
473
474 df_final = pd.DataFrame(df_final)  # makes a copy of the df
475 df_out_col = df_final.columns.values.tolist()
476 df_out_col_bin = [v for v in df_out_col if v == str(entity_id_col_m) or v not in cols_to_update]
477 df_bins = df_final[df_out_col_bin]
478
479 # Full outer join produces the set of all records in Table A and Table B, with matching records from both sides
    where
480 # available. If there is no match, the missing side will contain null.
481 df_final = pd.merge(df_spe_info, df_bins, on=str(entity_id_col_m), how='outer')
482 out_col = check_final_col_order(df_current.columns.values.tolist(), cols_to_update)
483 df_final = df_final.reindex(columns=out_col)
484
485 # Step 7: Updated lead agency code to agency abbreviations; add in flags for new species, species without a
    range, and
486 # species removed from master list
```

```
487
488 df_final.loc[df_final[lead_agency_col] >= 2, lead_agency_col] = 'NMFS'
489 df_final.loc[df_final[lead_agency_col] == 1, lead_agency_col] = 'USFWS'
490 df_final[huc2_col_c].fillna('No Range file', inplace=True)
491 df_final.loc[df_final[huc2_col_c] == 'No Range file', ['Updated']] = 'No Range file'
492 df_archived = df_current.loc[~df_current[str(entity_id_col_c)].isin(df_spe_info[str(entity_id_col_c)])]
493 for s in df_archived[str(entity_id_col_c)].values.tolist():
494     ent_index = out_col.index(entity_id_col_m)
495     col_flag_removed = out_col[ent_index + 1]
496     df_final.loc[df_final[str(entity_id_col_m)] == s, [col_flag_removed]] = 'Species removed from master list-
    see ' \
497                                                 'DroppedSpecies table for species name'
498 # Step 8: Exports data frame to csv
499 # UpdatedBins_[date].csv'  # working bin tables with updates
500 # DroppedSpecies_[date].csv  # Species removed from master list
501
502 df_final.to_csv(outfile, encoding='utf-8')
503 df_archived.to_csv(archived_file, encoding='utf-8')
504 # Elapsed time
505 end_script = datetime.datetime.now()
506 print "Elapse time {0}".format(end_script - start_time)
507
```