Webinar Series

Data Use Skills
Featuring Data from Natural History Collections

September 21 - November 30, 2022

https://www.idigbio.org/content/biotaphy-2022-webinar-series
iDigBio: https://www.idigbio.org/content/biotaphy-2022-webinar-series

BiotaPhy 2022 Webinar Series

Add this event to your calendar:

Tue, 09/13/2022 - 9:03am -- jegoodwin

Do you want to learn how to use occurrence data and available software to address questions in ecology and evolutionary biology but haven’t had a chance to take a course and are overwhelmed by the options for self-teaching? Would you like to incorporate this sort of research into your classes but don’t have time to create the materials and examples? If either of these applies, then join the BiotaPhy Project for an upcoming series of 10 webinars designed to take you from biological question to data acquisition and cleaning to analysis and interpretation!

BiotaPhy is a collaboration among iDigBio, LifeMapper, and the Open Tree of Life. Learn more about the project and register for the webinar series here.

Webinars will be held Wednesdays at 12:30 pm Eastern time, beginning September 21. All webinars will be recorded and made available, so you won’t get behind if you need to miss a session. The schedule is:

- **Webinar 0:** Terms, Concepts, Data Formats – A Tutorial for Background
  Click here for PDF

- **Webinar 1:** Introduction: Scope and Research Potential for Multidisciplinary Biodiversity Modeling and Analysis
  Date: 09/21/2022
Thank You

Maria Cortez
Aimee Stewart
Jill Goodwin
Gil Nelson
Webinar 3

Clean Your Dirty Data
1. Species list must contain: ACCEPTED, UNIQUE and CONSISTENT names!

2. Manual ways of treating nomenclature: useful for small datasets and to subset large datasets.

3. Automated ways of treating nomenclature: useful for large datasets.

4. BiotaPhy framework is a great toolkit for automated pathway!
Introduce the importance of cleaning data before processing biodiversity analyses and practice a few ways to do so
Biological Objectives:

✓ How can we use occurrence data, and where can we find them?

✓ Introduce why cleaning data is an essential step to ensure biodiversity analyses are as sound as possible.
Technical Objectives:

- How to download occurrence data from iDigBio (and GBIF)
- How to clean large data sets effortlessly, consistently and rapidly using industrial-strength tools
Webinar organization

1. Exploring Concepts: How can we use occurrence data, and why is cleaning data an important step?
2. Demonstration: How to download data from iDigBio
3. On Your Own: More options for data downloads
4. Exercises: Using Biotaphy Tools for cleaning occurrence data
5. Session Summary, Q&A and Discussion
How can we use occurrence data, and where can we find them?
Occurrence data are essential for producing species distribution models, estimating phylogenetic diversity, and more!!

**Heuchera mexicana**  W. Schaffn. ex Small & Rydb., 1905

From Computarización del Herbario ENCB, IPN. Fase IV. Base de datos de la familia Pinaceae y de distintas familias de la clase Magnoliosida depositadas en el Herbario de la Escuela Nacional de Ciencias Biológicas, IPN

<table>
<thead>
<tr>
<th>Continent</th>
<th>North America</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country</td>
<td>Mexico</td>
</tr>
<tr>
<td>State/Province</td>
<td>Mexico</td>
</tr>
<tr>
<td>County/Parish</td>
<td>Ecatezingo</td>
</tr>
<tr>
<td>Locality</td>
<td>Campamemento Tlalomas, Ameameca</td>
</tr>
</tbody>
</table>

**Latitude:** 19.065  
**Longitude:** -98.6336111
Exploring Concepts

Species distribution models

Prunus geniculata

Phylogenetic diversity

now

2050

Proportional Phylogenetic Diversity

min= 0.16  max= 0.64

BiotaPhy
Where do we get occurrence data?
Where do we get occurrence data?

GBIF

iDigBio
Downloading data from portals: iDigBio

Making data and images of millions of biological specimens available on the web

Digitization
Learn, share and develop best practices

Sharing Collections
Documentation on data ingestion

Working Groups
Join in, contribute, be part of the community

Proposals
New tool and workshop ideas

Citizen Scientists
How can you help biological collections?
Downloading data from portals: iDigBio

Downloading directly from iDigBio website: idigbio.org
On your own: Downloading data: iDigBio

Download using iDigBio’s API (Application Programming Interface)
On your own: Downloading data: iDigBio

Download occurrence data from iDigBio

To download from iDigBio, full instructions are at the Download API reference.

To pull data from the command prompt, use the `curl` command to pull text response directly to terminal with the example query_url:

Euphorbia

```bash
$ curl https://api.idigbio.org/v2/download/?rq=%7B%22genus%22%3A%22euphorbia%22%7D&email=donotreply%40idigbio.org
```

[58979]

astewart@murderbot:~/git/tutorials$ {
"complete": false,
"created": "2022-05-02T15:28:41.739968+00:00",
"expires": "2022-06-01T15:28:41.628063+00:00",
"hash": "18911492e8517926cb3695f49f971cf065107016",
"query": {
"core_source": "indekterms",
"core_type": "records",
"form": "dwca-csv",
"mediarecord_fields": null,
"mq": null,
"record_fields": null,
"rq": {
"genus": "euphorbia"
}
},
"status_url": "https://api.idigbio.org/v2/download/d54c8ad7-6697-4096-9f11-b2a9a6041a38",
"task_status": "PENDING"
}
On your own: Downloading data: iDigBio

Then use `curl` on the resulting `status_url` field:

```
$ curl https://api.idigbio.org/v2/download/d54c0ad7-6697-4896-9f11-b2a9a6041a38
{
   "complete": false,
   "created": "2022-05-02T15:28:41.730968+00:00",
   "expires": "2022-06-01T15:28:41.735029+00:00",
   "hash": "18911492e8517926cb8693fc9f971cf066107016",
   "query": {
      "core_source": "indexesterm",
      "core_type": "records",
      "form": "dwc-csv",
      "mediarecord_fields": null,
      "mq": null,
      "record_fields": null,
      "rq": {
         "genus": "euphorbia"
      }
   },
   "status_url": "https://api.idigbio.org/v2/download/d54c0ad7-6697-4896-9f11-b2a9a6041a38",
   "task_status": "PENDING"
}
```
When the task_status shows “SUCCESS”:

```
$ curl https://api.idigbio.org/v2/download/d54c8ad7-6697-4096-9f11-b2a9a6041a38
{
  "complete": true,
  "created": "2022-05-02T15:28:41.730968+00:00",
  "download_url": "http://s.idigbio.org/idigbio-downloads/d54c8ad7-6697-4096-9f11-b2a9a6041a38.zip",
  "expires": "2022-06-01T15:28:41.552351-00:00",
  "hash": "18011492e8517926cb8693fc9f971cf066107916",
  "query": {
    "core_source": "indexterms",
    "core_type": "records",
    "form": "dwca-csv",
    "mediarecord_fields": null,
    "mq": null,
    "record_fields": null,
    "rq": {
      "genus": "euphorbia"
    }
  },
  "status_url": "https://api.idigbio.org/v2/download/d54c8ad7-6697-4096-9f11-b2a9a6041a38",
  "task_status": "SUCCESS"
}
```
On your own: Downloading data: iDigBio

When the task_status shows “SUCCESS”:

```bash
$ curl https://api.idigbio.org/v2/download/d54c0ad7-6697-4096-9f11-b2a9a6041a38
{
  "complete": "true",
  "created": "2022-05-02T15:28:41.730968+00:00",
  "download_url": "http://s.idigbio.org/idigbio-downloads/d54c0ad7-6697-4096-9f11-b2a9a6041a38.zip",
  "expires": "2022-06-01T15:28:41.552351+00:00",
  "hash": "18911492e8517926cb8693fc9f971cf066107016",
  "query": {
    "core_source": "indexterms",
    "core_type": "records",
    "form": "dwca-csv",
    "mediarecord_fields": null,
    "mq": null,
    "record_fields": null,
    "rq": {
      "genus": "euphorbia"
    },
    "status_url": "https://api.idigbio.org/v2/download/d54c0ad7-6697-4096-9f11-b2a9a6041a38",
    "task_status": "SUCCESS"
  }
}
```

Save the response into a file with the `wget` command and the `download_url` field:

```bash
wget http://s.idigbio.org/idigbio-downloads/d54c0ad7-6697-4096-9f11-b2a9a6041a38.zip
```
On your own: Downloading data: GBIF
On your own: Downloading data: GBIF
On your own: Downloading data: iDigBio & GBIF

- ridigbio
- rgbif
- spocc: Interface to Species Occurrence Data Sources
Exploring Concepts

Why is cleaning data an important step, and why and how do we clean our data?
Exploring Concepts

Occurrence Records:

• Specimen information: typically reproduced directly from labels
• Aggregators do only minimal or no cleaning
• Errors may result
• May need to modify set of records
Examples of Data Cleaning Issues:

- Resolve taxon names
- Remove duplicates (physical & electronic)
- Clean localities
  - Round up the latitude/longitude
  - Remove coordinates at 0,0
  - Remove coordinates in cultivated zones, botanical gardens, etc.
- Remove coordinates outside of the desired range
- Spatial correction to optimize data set for modeling
Examples of Data Cleaning Issues:

- Resolve taxon names
- Remove duplicates (physical & electronic)
- Clean localities
  - Round up the latitude/longitude
  - Remove coordinates at 0, 0
  - Remove coordinates in cultivated zones, botanical gardens, etc.
  - Remove coordinates outside of the desired range
- Spatial correction to optimize data set for modeling
Examples of Data Cleaning Issues:

• Resolve taxon names
• Remove duplicates
• Clean localities
  • Round up the latitude/longitude
  • Remove coordinates at 0,0
  • Remove coordinates in cultivated zones, botanical gardens, etc.
• Remove coordinates outside of the desired range
• Spatial correction to optimize data set for modeling
Examples of Data Cleaning Issues:

✓ Resolve taxon names
✓ Remove duplicates
• Clean localities
  • Round up the latitude/longitude
  • Remove coordinates at 0,0
  • Remove coordinates in cultivated zones, botanical gardens, etc.
• Remove coordinates outside of the desired range
• Spatial correction to optimize data set for modeling
Examples of Data Cleaning Issues:
- Resolve taxon names
- Remove duplicates
- Clean localities
  - Round up the latitude/longitude
  - Remove coordinates at 0,0
  - Remove coordinates in cultivated zones, botanical gardens, etc.
  - Remove coordinates outside of the desired range
- Spatial correction to optimize data set for modeling
What happens when there is a large dataset?

Should we manually clean thousands to millions of records?

We use BiotaPhy tools to automate cleaning data!
Let’s put the automated framework developed by BiotaPhy to test!

How to clean your data:

3 steps:

✓ Data Preparation
✓ Run Tutorial
✓ Inspect Output

Input: Occurrence records
Input: Wrangler configuration files
Input: Script parameter file
Before we start ...

Download for the first time OR update the tutorials repository containing test data and configurations.

**Initial download:**

```
git clone https://github.com/biotaphy/tutorials
```

**Update tutorial:**

```
cd tutorials
git pull
```
Let’s rebuild our Docker images to incorporate any updates. Move to the directory containing the tutorials repository that you downloaded or updated.

- Remove old docker elements: `./run_tutorial.sh cleanup_all`
- Rebuild data and image: `./run_tutorial.sh build_all`

Windows users will run with: `run_tutorial.bat`
**Data preparation**

**Input: occurrence records**

The `wrangle_occurrences` tool accepts either a Darwin Core Archive (DwCA) file or a CSV file containing records for one or more taxa. More information is in the Occurrence Data section of Specimen Occurrences: Data and Wrangling.

<table>
<thead>
<tr>
<th>species_name</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bensoniola oregona</td>
<td>-123.751</td>
<td>42.802</td>
</tr>
<tr>
<td>Bensoniola oregona</td>
<td>-123.7903</td>
<td>42.802</td>
</tr>
<tr>
<td>Bensoniola oregona</td>
<td>-123.7903</td>
<td>42.802</td>
</tr>
<tr>
<td>Bensoniola oregona</td>
<td>-123.7707</td>
<td>42.7873</td>
</tr>
<tr>
<td>Bensoniola oregona</td>
<td>-123.751</td>
<td>42.9927</td>
</tr>
<tr>
<td>Bensoniola oregona</td>
<td>-123.9646</td>
<td>42.7788</td>
</tr>
<tr>
<td>Bensoniola oregona</td>
<td>-123.7117</td>
<td>42.9047</td>
</tr>
<tr>
<td>Bensoniola oregona</td>
<td>-123.7117</td>
<td>42.9047</td>
</tr>
<tr>
<td>Bensoniola oregona</td>
<td>-123.8266667</td>
<td>42.625</td>
</tr>
</tbody>
</table>
### Data Preparation: occurrence records

#### Structure of a CSV file containing occurrence records

<table>
<thead>
<tr>
<th>Species name</th>
<th>Longitude</th>
<th>Latitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bensoniella oregona</td>
<td>123.751</td>
<td>42.802</td>
</tr>
<tr>
<td>Bensoniella oregona</td>
<td>123.7903</td>
<td>42.802</td>
</tr>
<tr>
<td>Bensoniella oregona</td>
<td>123.7903</td>
<td>42.802</td>
</tr>
<tr>
<td>Bensoniella oregona</td>
<td>123.7707</td>
<td>42.7873</td>
</tr>
<tr>
<td>Bensoniella oregona</td>
<td>123.751</td>
<td>42.9927</td>
</tr>
<tr>
<td>Bensoniella oregona</td>
<td>123.9646</td>
<td>42.7788</td>
</tr>
<tr>
<td>Bensoniella oregona</td>
<td>123.7117</td>
<td>42.9047</td>
</tr>
<tr>
<td>Bensoniella oregona</td>
<td>123.7117</td>
<td>42.9047</td>
</tr>
<tr>
<td>Bensoniella oregona</td>
<td>123.826667</td>
<td>42.625</td>
</tr>
</tbody>
</table>
Input: Wrangler configuration file

The tool allows multiple operations, defined in a wrangler configuration file, to be applied to the data at the same time. A data wrangler configuration is a file containing a JSON list of zero or more wranglers - each performs a different operation, and each has its own parameters. The file is specified in the Script parameter file described above.

More information on file format, available wrangler types, and the required and/or optional parameters for each are in the Occurrence Wrangler Types section of data_wrangle_occurrence.

In this example, we will resolve occurrence data names with GBIF using the AcceptedNameOccurrenceWrangler and also the DecimalPrecisionWrangler, which filters out points without a certain number of digits past the decimal point. Our wrangler configuration file occ_wrangle_resolve.json contains these parameters.
Data Preparation: wrangler file

Input: Wrangler configuration file

In this example, we will resolve occurrence data names with GBIF using the AcceptedNameOccurrenceWrangler and also the DecimalPrecisionWrangler, which filters out points without a certain number of digits past the decimal point. Our wrangler configuration file `occ_wrangle_resolve.json` contains these parameters.

```json
[  
    {  
      "wrangler_type": "AcceptedNameOccurrenceWrangler",
      "name_resolver": "gbif",
      "out_map_filename": "/volumes/output/occ_wrangle_resolve.namemap",
      "map_write_interval": 100,
      "out_map_format": "json"
    },
    {  
      "wrangler_type": "DecimalPrecisionFilter",
      "decimal_places": 4
    }
  ]
```
Data Preparation: wrangler file

Different types of wranglers

- AcceptedNameOccurrenceWrangler
- AttributeFilterWrangler
- AttributeModifierWrangler
- BoundingBoxFilter
- CommonFormatWrangler
- CoordinateConverterWrangler
- DecimalPrecisionFilter
- DisjointGeometriesFilter
- IntersectGeometriesFilter
- MinimumPointsWrangler
- SpatialIndexFilter
- UniqueLocalitiesFilter
Data Preparation: wrangler file

**AcceptedNameOccurrenceWrangler**

The AcceptedNameOccurrenceWrangler matches the value in the occurrence data identified as the "species" field with an "accepted name" as defined in a name-map or by a taxonomic service.

- **optional**
  - **name_map (str or dict):** A dictionary or filename containing a dictionary of original name to accepted name. Defaults to None, but either this or name_resolver must be provided.
  - **name_resolver (str or Method):** Use this method for getting new accepted names. If set to 'gbif' or 'otol', use GBIF or OTOL name resolution respectively. Defaults to None, but either this or name_map must be provided.
  - **out_map_filename (str):** Output for name-mapping between original and accepted names. This file is then acceptable for use as a name-map input for subsequent name wrangling. Defaults to None.
  - **map_write_interval (int):** Interval at which to write records to disk. Used to ensure that if something fails, all is not lost. Defaults to 100.
  - **out_map_format (str):** Type of file format for out_map_filename, defaults to "json".
  - **store_original_attribute (str):** A new attribute to store the original taxon name.
AttributeFilterWrangler

The AttributeFilterWrangler filters out points based on whether the value in the given attribute passes the given function.

- required
  - attribute_name (str): The name of the attribute to modify.
  - filter_func (Method): A function to be used for the pass condition.
AttributeModifierWrangler

The AttributeModifierWrangler modifies a newly added or existing attribute, computing the value with the given function.

- required
  - attribute_name (str): The name of the attribute to modify.
  - attribute_func (Method): A function to generate values for a point.
The BoundingBoxFilter filters out occurrence points if they do not fall within the given bounding box.

- **required**
  - `min_x` (numeric): The minimum ‘x’ value for the bounding box.
  - `min_y` (numeric): The minimum ‘y’ value for the bounding box.
  - `max_x` (numeric): The maximum ‘x’ value for the bounding box.
  - `max_y` (numeric): The maximum ‘y’ value for the bounding box.
The CommonFormatWrangler modifies points to a common format, using the given attribute-map between the original fields, and the desired fields in the common format.

- required
  - attribute_map (dict): A mapping of source key, target values.
CoordinateConverterWrangler

The CoordinateConverterWrangler modifies occurrence points by transforming the x and y coordinates from one projection (coded as an EPSG number) into another projection. The new coordinates overwrite the x and y fields. If original_x_attribute and original_y_attribute are provided, these should be new fields in which to save the original x and y coordinates.

- required
  - target_epsg (int): Target map projection specified by EPSG code.
- optional
  - source_epsg (int): Source map projection specified by EPSG code. Either this or epsg_attribute MUST be provided.
  - epsg_attribute (str or None): A point attribute containing EPSG code. Either this or source_epsg MUST be provided.
  - original_x_attribute (str): An attribute to store the original x value.
  - original_y_attribute (str): An attribute to store the original y value.
The `DecimalPrecisionFilter` filters out occurrence points where one or both coordinates have values where the number of digits to the right of the decimal point is less than the given number.

- required:
  - `decimal_places` (int): Only keep points with at least this many decimal places of precision.
**DisjointGeometriesFilter**

The DisjointGeometriesFilter filters out points where the coordinates intersect with the given geometries.

- **required:**
  - `geometry_wkts` (list of str): A list of geometry WKTs to check against.
**IntersectGeometriesFilter**

The IntersectGeometriesFilter filters out points where the coordinates do NOT intersect with the given geometries.

- **required:**
  - `geometry_wkts` (list of str): A list of WKT strings.
MinimumPointsWrangler

See the Note above for important information on the use of this wrangler.

The MinimumPointsWrangler filters out groups of points where the number of points in a group does not meet the minimum.

- required:
  - minimum_count (int): The minimum number of points in order to keep all.
**SpatialIndexFilter**

The SpatialIndexFilter filters out points that match some given condition (check_hit_function) on the given spatial index.

- required:
  - `spatial_index` (SpatialIndex): A SpatialIndex object that can be searched.
  - `intersections_map` (dict): A dictionary of species name keys and corresponding valid intersection values.
  - `check_hit_func` (Method): A function that takes two arguments (search hit, valid intersections for a species) and returns a boolean indication if the hit should be counted.
Data Preparation: wrangler file

UniqueLocalitiesFilter

See the Note above for important information on the use of this wrangler.

The UniqueLocalitiesFilter filters out points from a grouping that do not have unique coordinates. The filter can operate on one or more groups, and uniqueness is only checked within groups.

- optional parameters:
  - do_reset (bool): Reset the list of seen localities after each group.
Data Preparation: script file

Input: Script parameter file

A JSON parameter file is required for this command. The tutorial parameter file is `wrangle_occurrences_resolve.json`. These are the required and optional parameters:

- Required:
  - `reader_filename`: The input CSV file of occurrence records.
  - `writer_filename`: A file location to write cleaned points.
  - `wrangler_configuration_file`: occurrence wrangler configuration file, described in the next section. The tutorial example wrangler configuration contains several wranglers, the DecimalPrecisionFilter, the UniqueLocalitiesFilter, MinimumPointsWrangler, and the AcceptedNameOccurrenceWrangler, and is in `occ_wrangle_resolve.json`

Choose one or more of the wranglers specified previously!

Reminder

- `Reader_filename = Input`
- `Writer_filename = Output`
Data Preparation: script file

- **species_key**: The field name of the input file column containing species data. The default value is *species_name*, so if the data contains any other column name for the field to group on, this must be specified.
- **x_key**: The field name of the input file column containing x/longitude coordinate. The default value is x, so if the data contains any other column name for the x coordinate, this must be specified.
- **y_key**: The field name of the input file column containing y/latitude coordinate. The default value is y, so if the data contains any other column name for the y coordinate, this must be specified.
- **geopoint**: The field name of the input file column containing a JSON encoded geopoint (IDigBio data uses this field), with sub-elements containing the x and y keys with their coordinates. The default value is None.
- **log_filename**: Output filename to write logging data
- **log_console**: ‘true’ to write log to console
- **report_filename**: Output filename with data modifications made by wranglers

- **Species names**
- **Longitude**
- **Latitude**
- **Log filename**
- **Logging appears on console**
- **Modifications done by wrangler**
Input: Script parameter file

A JSON parameter file is required for this command. The tutorial parameter file is `wrangle_occurrences_resolve.json`. These are the required and optional parameters:

```json
{
  "species_key": "species_name",
  "x_key": "x",
  "y_key": "y",
  "report_filename": "/volumes/output/wrangle_occurrences_w_resolve_report.json",
  "log_filename": "/volumes/output/wrangle_occurrences_w_resolve.log",
  "log_console": true,
  "reader_filename": "/volumes/data/input/heuchera.csv",
  "writer_filename": "/volumes/output/heuchera_clean_accepted.csv",
  "wrangler_config_filename": "/volumes/data/wranglers/occ_wranglers_w_resolve.json"
}
```
Let's run this tutorial!

Goal: produce a file containing edited occurrence data according to the wrangler(s) used!

Run tutorial

Initiate the process with the following:

```
#!/bin/bash
./run_tutorial.sh wrangle_occurrences data/config/wrangle_occurrences_resolve.json
```

Windows users will run with: `run_tutorial.bat`

Remember, you will RUN this code in the terminal or in the command prompt for Windows!
Let’s look at the output!

Output

This process outputs files configured in the script parameter file:

2. If `report_filename` is specified in the script parameter file, a summary of name resolutions, like

```bash
wrangle_occurrences.log
```

<table>
<thead>
<tr>
<th>Author</th>
<th>Commit</th>
<th>Date</th>
<th>Time</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>zzeppozz</td>
<td>e696df</td>
<td>09 Sep 2022</td>
<td>15:14</td>
<td>Added code for filtering and assessing points.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lines</th>
<th>SLOC</th>
<th>Size</th>
<th>Raw</th>
<th>Blame</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>6224</td>
<td>6224</td>
<td>698</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. AcceptedNameOccurrenceWrangler: Filtered 0, modified 0 of 0 assessed points.
2. DecimalPrecisionFilter: Filtered 0, modified 0 of 0 assessed points.
3. AcceptedNameOccurrenceWrangler: Resolution identical: Bensoniella oregona
4. AcceptedNameOccurrenceWrangler: Return 51 of 51 points, total of 0 modified
5. wrangle_occurrences: AcceptedNameOccurrenceWrangler wrangled 51 from /Volumes/data/input/heuchera.csv
6. DecimalPrecisionFilter: Bensoniella oregona -123.751, 42.802 fails precision test.
7. DecimalPrecisionFilter: Bensoniella oregona -123.7903, 42.802 fails precision test.
8. DecimalPrecisionFilter: Bensoniella oregona -123.7903, 42.802 fails precision test.
Let’s look at the output!

3. If `log_filename` is specified in the script parameter file, a report like `wrangle_occurrences.rpt` containing a summary of the processing.

4. If `log.console` is specified in the script parameter file, logs will be written to the command prompt during execution.

```json
{
    "input_records": 11409,
    "output_records": 2203,
    "wranglers": [
        {
            "name": "AcceptedNameOccurrenceWrangler",
            "version": "1.0",
            "assessed": 0,
            "modified": 0,
            "filtered": 0
        },
        {
            "name": "DecimalPrecisionFilter",
            "version": "1.0",
            "assessed": 0,
            "modified": 0,
            "filtered": 0
        }
    ]
}
```
Let’s look at the output!

4. an output file with occurrence records named in the writer_filename, like
   heuchera_wrangled.csv containing the occurrence records, one record per line. Note that the
   species_name field now contains the new taxonomic name resolved for each record. If the
   original records contain other attributes, those will be retained with their original values.

If the wrangle configuration file contains the AcceptedNameOccurrenceWrangler, as in the
command above, using the wrangle_occurrences_resolve.json configuration file, the process
produces one additional file as configured in that wrangler configuration:

<table>
<thead>
<tr>
<th>species_name</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bensoniella oregona</td>
<td>-123.7707</td>
<td>42.7873</td>
</tr>
<tr>
<td>Bensoniella oregona</td>
<td>-123.9646</td>
<td>42.7788</td>
</tr>
<tr>
<td>Bensoniella oregona</td>
<td>-123.7117</td>
<td>42.9047</td>
</tr>
<tr>
<td>Bensoniella oregona</td>
<td>-123.7117</td>
<td>42.9047</td>
</tr>
<tr>
<td>Bensoniella oregona</td>
<td>-123.803889</td>
<td>42.60861111</td>
</tr>
</tbody>
</table>

Webinar 3
Let’s look at the output!

If the wrangler configuration file contains the AcceptedNameOccurrenceWrangler, as in the command above, using the wrangle_occurrences_resolve.json configuration file, the process produces one additional file as configured in that wrangler configuration:

- An out_map_filename containing a name-map from the AcceptedNameOccurrenceWrangler. The name-map is a JSON file with pairs of names - the original name to the accepted name according to the specified authority. This name-map is suitable to use for input when resolving another dataset containing a subset of the same original names. A sample output name-map is occ_wrangle_resolve.namemap.

```json
1 { 
 2   "Bensiella oregona": "Bensiella oregona",
 3   "Conimitella williamsii": "Conimitella williamsii",
 4   "Elmera racemosa": "Elmera racemosa",
 5   "Heuchera abramsii": "Heuchera abramsii",
 6   "Heuchera acutifolia": "Heuchera acutifolia",
 7   "Heuchera alba": "Heuchera alba",
 8   "Heuchera americana": "Heuchera americana",
 9   "Heuchera bracteata": "Heuchera bracteata",
10   "Heuchera brevistaminea": "Heuchera brevistaminea",
```
Multiple ways to obtain occurrence records:
  iDigBio, GBIF, other aggregators
Portals, APIs, R packages
Data can be very dirty!
  errors in names, spelling, locality descriptions, GPS coordinates, more!
Duplicates (physical & electronic)
May need to optimize (remove certain areas, reduce density of samples, etc.)
Save your clean data and consider making data set available online!
Any questions??

Please use the Chat to ask your question!