**Final writeup with these parts:**

Title: A short title indicating the topic.

Synopsis: up to 4 sentences telling what your project does. (you may reuse what you submitted

for the project gallery.)

Extended abstract: (<=300 words) The abstract should give a very clear brief high-level

description of the problem you're addressing and how you solve it. Assume the audience is not an expert in your subject area. Use a spelling-checker! See examples.

Pseudocode: Pseudocode for the problem described in the previous step. Use pseudocode

keywords and keep the level of detail broad.

Acknowledgments: Acknowledge help received writing the code or any scripts you used which were authored by others (aside from course examples)

Keywords: Comma-separated list of keywords relating to the project. These should include topic areas, data types, and tools. This is meant to help other students search for related project topics.

Mapping: One or two sentences explaining how you used the mapping module and what is automatically added to the map.

GUI: A screenshot of your graphical user interface and a table with a list of parameters (should have the same columns as in the revised proposal: Parameter, Data Type, Type (Required, optional, or derived), Direction, Filter. See the extended proposal instructions for an example.)

**Final presentation with two parts:**

Presentation 1: Explain the project(<= 5 minutes)

* File should be named: unityID\_Pres1
* Describe the problem the code solves (screen recording with voiceover). Be clear, concise, and creative
* Include a checklist of the required components (see Code Evaluation part I below). List how the requirement was met --or why it was not.
* If applicable, note any advanced functionality Python that you used beyond the requirements or requiring the study of Python functionality beyond the course material.

Presentation 2: Prerecorded demo (<=5 minutes)

* File should be named: unityID\_Pres2
* Introduce the project topic very briefly--give the project title and the goal of the project in 1-3 sentences (and a picture, if possible).
* Demonstrate how to run the code. During this video, run the code, starting with the GUI. Make sure to show the button (if applicable), the interface, and the results being added automatically to the map.
* Tell us about any extra functionality you've learned and implemented. This is your chance to brag.
* Both distance and traditional students pre-record a video. Loom is recommended for free video screen capture.

**Code requirements:**

Map files, scripts, toolboxes, test data any other supporting files needed to test the code. Test data should not be massive. If the data is massive, please submit only a small dataset. Please indicate any privacy restrictions on the data, otherwise, it may be made publicly available.

Set default values in the graphical user interfaces so that we can run your code without assistance.

Check for portability (if I move it to another machine, does it break?)

Implement instructor feedback from proposal (and extended proposal) submissions.

**Code components:**

Category I (Complete all of a-f for this category)

Python geoprocessing (call arcpy tools).

Python batch processing (FOR or WHILE loops)

Code reuse: Define and call at least 3 user-defined reusable functions. Remember not

to use functions presented as examples and not to create functions that are simply

wrappers for geoprocessing tools (opens in new window).

Code reuse: Define a class and instantiate and use an object of this type.

Mapping: Automatically display geoprocessing output in the map. (If the output is

better viewed outside of ArcGIS, you must still display it automatically. But also show

messages so that the user knows how to find and display the visual output in a more

appropriate viewer.)

GUI: Use Python script tools and script tool properties to generate a graphical

interface.

Category II (Pick one or more)

Reading and/or writing files with standard file objects or with cursors.

Reading and/or writing HTML to automatically generate an output report.

Reading and/or writing KML with BeautifulSoup.

Downloading and uncompressing data from the web.

Category III (Pick one or more--these are easier and don't carry as much weight as Cat II)

Use the tool validator to create dynamic behavior for the script tool.

Provide a portable toolbar button to launch the script tool.(Not supported in Pro.)

Inform the user of progress, using messages.

Add the arcpy progressor to present messages.

In the script tool, use the parameter filter property, the parameter "obtained from"

property, the parameter symbology property, or the Schema property.

**Programming style:**

Provide a valid usage example in the header comments.

Use of variables instead of repeating string literals;

Minimize hard-coding. File paths should not be hard-coded.

Multiple lines of code that are invoked more than once with only slight variation should

be put into a function with the variations passed in as parameters.

Use conditional expressions efficiently (e.g., if foo.endswith("bla") instead of if foo.endswith("bla") == True)

Use conditional blocks efficiently (e.g., use elif/else when they can be used instead of sequential if/if/if)

Only import modules and packages that are being used.

Use of data structures such as lists/dictionaries to streamline code

Lightly commented but also self-documenting (easy to guess what's happening)

Use modules to group tightly related functions to facilitate reusability.

Avoid the use of hard-coded values

Catch exceptions to avoid traceback exceptions

Lines of code <=90-ish characters each.

Use a docstring inside the start of each function and inside the start of each class.

Benton Tripp

GIS 540 Final Project Writeup

1. **Instructor Feedback:** I only received one suggestion for my project. It was suggested that I test out different hyperparameters for each species when modeling their respective distributions using the MaxEnt algorithm (i.e., use batch processing to test out different parameters and keep the best for each species). This is a fairly simple change on my part. I have not yet implemented it into my code, but I will just add some additional arguments to the batchMaxEnt function to account for the different parameters that will be tested (this could further be incorporated into an additional script tool if time permits).
2. **Title:**North Carolina Woodpecker Species Distribution Modeling using Presence-Only Prediction
3. **Abstract:** A major challenge in understanding the distribution of species is that observational point data tends to be primarily recorded in locations easily accessible to people. One method of estimating the distribution in a broader area is through a spatial statistics technique called Presence Only Prediction using Maximum Entropy (MaxEnt)1. The purpose of this project is to use the MaxEnt approach to estimate woodpecker species’ distributions in North Carolina from crowd-sourced bird watching dataset.
4. **Input data:** The data used in this project are from three primary sources:
   1. Project FeederWatch2 – Bird observation data, compiled by The Cornell Lab of Ornithology and Birds Canada. There are two file formats for the data: (1) *.xlsx* and (2) *.csv*. The Excel file contains the “data dictionary”, and is primarily needed so that the common names of the woodpecker species can be joined with the *.csv* data (in which they are instead identified by a unique ID). The *.csv* data contains the relevant data needed for this project including the species, date observed, and the location observed. The data can be obtained directly from the Project FeederWatch website via the dataset requests page: <https://feederwatch.org/explore/raw-dataset-requests/>.
   2. National Land Cover Database3 – using data from the Multi-Resolution Land Characteristics (MRLC) Consortium4, North Carolina State University created the NLCD. The data used in this project is clipped/re-projected land cover data for the state of North Carolina. <https://gisdata.lib.ncsu.edu/fedgov/mrlc/nlcd2019/NC_NLCD2019only.zip>
   3. North Carolina Digital Elevation Model (DEM)5 – this data was acquired directly through the NCSU GIS library at: <https://gisdata.lib.ncsu.edu/DEM/nc250.zip>. The data is a raster image of North Carolina elevation at a 1:250,000 scale.

The observation data will serve as the input point features for the MaxEnt models (each species will be modeled separately), while the two raster datasets will be the explanatory rasters.

1. **Data Products:** The original bird observation data is sparse, and tends to be most prevalent in human-accessible areas where birdwatching is popular. The Presence-Only Prediction (MaxEnt) Algorithm will use explanatory variables (i.e., elevation and land cover) to estimate how likely it might be to observe the same birds across the entire study area. The function available through *arcpy* provides several outputs:

* Trained raster (*see Figure 1 of the example from the arcpy Presence-Only Prediction documentation*6)
* Trained features (background point data generated to aid in training the model against the explanatory features)
* Descriptive tables (response curve, sensitivity)

The primary output of concern in the scope of this project is the trained raster, since it can be used to easily visualize the estimated distribution of the input bird species across the study area.

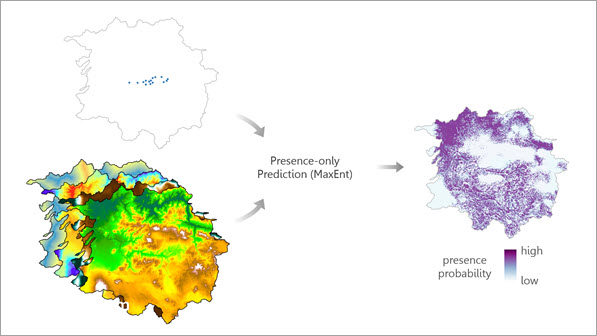
[](https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/presence-only-prediction.htm)

Figure 1: Point data, explanatory raster(s), and other explanatory variables are used as inputs to the Presence-Only Prediction (MaxEnt) algorithm, and a presence probability raster is output for the defined study area. This image was taken from the arcpy Presence-Only Prediction documentation6

1. **Revised pseudocode (updates in bold):**

FUNC get\_species\_codes(species codes url, bird family)

GET species codes from species codes url

COMPUTE filtered species codes by filtering the data

by bird family

RETURN filtered species codes

END FUNC

FUNC clean\_data(raw feeder watch data, filtered species, US

state, minimum year, year)

SET list of all of the correct names

FOR name in raw feeder watch data field names

IF name not in correct names

Rename to the correct name

END IF

END FOR

SET filtered feeder watch data as the raw feeder

watch data filtered by the specified species

SET filtered feeder watch data as the filtered feeder

watch data filtered again, but the specified US

state

SET filtered feeder watch data as the filtered feeder

watch data filtered again, but as greater than or

equal to the minimum year, less than or equal to

the maximum year

SET final feeder watch data as the filtered feeder

watch data where the valid field is set to True

RETURN final feeder watch data

END FUNC

FUNC get\_feeder\_watch\_data(outfile, time frame list,

filtered species, US state,

minimum year, maximum year,

output directory)

SET final output .csv file name

IF final output file exists THEN

READ feeder watch data from output file

ELSE

SET data list

FOR time frame in time frame list

SET intermediate output .csv file name

IF intermediate file does not exist

SET feeder watch data url for the time

frame

GET raw feeder watch data from url

cleaned data = clean\_data(raw feeder

watch data,

filtered

species, US state, minimum year, maximum year)

Save cleaned data to intermediate file

ELSE

READ intermediate file

ENDIF

Append cleaned data to data list

ENDFOR

SET feeder watch data as the combined data list

ENDIF

RETURN feeder watch data

END FUNC

FUNC batch\_db\_import(feeder watch .csv file name, feature

class name, existing feature classes,

projected coordinate system, output

directory, feeder watch data, filtered

species data)

IF base feature class name not in existing feature

classes THEN

Create a new feature class from the feeder watch

.csv file, located at the output directory

ENDIF

IF projected base feature class not in existing

feature classes THEN

Create a new feature class using the base feature class and the projected coordinate system

ENDIF

GET unique species names from feeder watch data

SET species feature class list as empty list

FOR species name in unique species names

SET species feature class name

IF species feature class name not in existing

feature classes THEN

Select the subset of the base feature class (with the projected coordinate system) containing the species name, and create a new feature class called the species feature class name

ENDIF

Append species feature class name to species

feature class name list

ENDFOR

RETURN species feature class list

END FUNC

FUNC get\_rasters(land cover raster url, elevation raster

url, land cover output path, elevation

raster output path, land cover raster

name, elevation raster name, land cover

resample size)

IF land cover output path does not exist THEN

GET land cover raster from land cover raster url

Save to land cover output path

Resample land cover data using land cover

resample size, save to geodatabase as land cover

raster name

ENDIF

IF elevation raster output path does not exist THEN

GET elevation raster from elevation raster url

Save to elevation raster output path

Copy elevation raster to geodatabase as elevation

raster name

ENDFUNC

**FUNC batch\_maxent(species dataframe, dictionary of**

**parameter names/value list pairs,**

**outfile names)**

**SET empty model list**

**FOR distinct species in species dataframe**

**FOR each parameter option**

**SET MaxEnt output file names**

**COMPUTE MaxEnt using the species feature class, elevation raster, and resampled land cover raster; outputs are saved to the defined MaxEnt output file names**

**ENDFOR**

**GET best model from computed models and append**

**models and model outputs to the model list**

**ENDFOR**

**RETURN model list**

**ENDFUNC**

SET feeder watch .csv file name

SET species codes url

SET bird family

filtered species data = get\_species\_codes(species codes

url, bird family)

SET time frame list

SET minimum year

SET maximum year

SET ouput directory

SET filename suffix

feeder watch data = get\_feeder\_watch\_data(feeder watch .csv

file name, time

frame list,

filtered species

data, US state,

minimum year,

maximum year,

ouput directory,

filename suffix)

SET project path

SET database name

IF database name not in project path THEN

Create the database

ENDIF

SET workspace as database path

GET existing feature classes (might be empty list)

SET projected coordinate system

SET base feature class name

species feature class list = batch\_db\_import(feeder watch

.csv file

name, base

feature class name, existing feature classes, projected coordinate system,

output

directory, feeder watch data, filtered species data)

SET land cover raster url

SET land cover output path

SET land cover raster name

SET land cover resample size

SET elevation raster url

SET elevation raster output path

SET elevation raster name

get\_rasters(land cover raster url, elevation raster url,

land cover output path, elevation raster output

path, land cover raster name, elevation raster

name, and cover resample size)

**models and model outputs = batch\_maxent(species dataframe,**

**dictionary of**

**parameter**

**names/value list**

**pairs, outfile**

**names)**

**FOR model output in model outputs**

**GET map template**

**ADD output to map template**

**ENDFOR**

1. **Keywords:**species distribution, raster, point data, digital elevation model, land cover, presence-only prediction, MaxEnt, statistics, birds
2. **Progress report:**

* The code is nearly finished. The current code (submitted with this report) is detailed/structured as follows:
  + *woodpeckers\_nc.py* – The primary script that can be run to conduct the full analysis (it might not be working at the moment, since some changes were made to the other scripts in order to accommodate for the *Download and Preprocess FeederWatch Data* tool)*.*
  + *data\_setup\_tool.py* – The script linked to the *Download and Preprocess FeederWatch Data* tool. Runs the data downloading/processing part of the analysis.
  + *get\_bird\_data.py* – Downloads and pre-processes the FeederWatch (bird) data.
  + *process\_bird\_data.py –* Called from *get\_bird\_data.py*; pre-processes the bird data.
  + *get\_explanatory\_data.py* – Downloads and pre-processes the landcover rasters and DEM data.
  + *birds.py* – Species and Bird classes (see Category I, part d. for more details).
  + *presence\_only.py –* Models bird species distribution using the MaxEnt Algorithm.
  + *\_\_init\_\_.py*
* The primary components remaining are:
  + Updates to *presence\_only.py*; As discussed in this report, I intend to add some additional logic that iterates through different parameters for each of the species.
  + Add an additional script for mapping model outputs.
  + Final polishes (Refactoring code, ensuring all necessary details included in docstrings and comments, making sure references contain all of the necessary information, etc.).

1. **Category I** 
   1. **Python geoprocessing:**One of the tools that will be used is the resampling (raster) tool: arcpy.management.Resample. Because the landcover rasters are very high-resolution, the MaxEnt algorithm is unable to handle the unprocessed rasters (it results in an error). Resampling helps to reduce the resolution while maintaining as much integrity of the data as possible.
   2. **Python batch processing (FOR or WHILE loops): “**Batch processing” is intermingled throughout the entire project. The basis of the project is to conduct an analysis across several different species of woodpecker, so each of the steps (data loading, pre-processing, and analysis) are conducted from within a for loop that for each of the species.

The processing problem will include:

1. Pulling in the FeederWatch data (since the data is spread across multiple .csv files by date, this will be done as a batch process). Each individual part of the data will be preprocessed before all of them are combined into a final dataset and saved to a .csv file.
   1. The preprocessing step includes:
      1. Filtering the data to only consist of bird species in the Woodpecker family
      2. Removing possible “bad” records (there is a field that specifies if the record was verified)
      3. Fixing field names
      4. Filter by US State
      5. Filtering by date range
2. Set up environment (i.e., create geodatabase, define projected coordinate system, etc.)
3. For each species in the final data, create a Feature Class and add the defined projected coordinate system to the feature class.
4. Pull in Raster Datasets, and save to the Geodatabase
5. Resample the Land Cover raster to be 1000 x 1000 meters
6. For each species of woodpecker, use the Presence-Only-Prediction tool to estimate the distribution of that species in the state of North Carolina.
   1. **Iterate through different hyperparameters for each species, and select the best respective models.**
   2. **Functions:** Below is a list of three of the primary functions used, along with the contents of their respective docstrings (each provide a brief description, and a list of inputs/outputs).

getFeedWatcherData: Gets FeederWatch data from website. When reading directly from the URLs and saving the output, this can take a while (depending on internet speed). Each independent query (by date range) is saved to a gzipped .csv file, so if the process is interrupted or re-run, it can be read directly from that file instead of re-downloading. Data is also cleaned/filtered (using clean\_fw\_data()), then concatenated and saved to a final .csv file.

Args:

* outfile: Final output file name
* tfs: Time-frames to get data for
* birds: Species data (Optionally) pre-filtered (e.g., by family)
* sub\_national\_code: (Optionally) filter by subnational1\_code (e.g., U.S. State)
* out\_dir: Directory in which to save data
* file\_suffix: Suffix of file names
* save\_: Whether or not to save the output to a gzipped file
* min\_year: minimum year to filter data
* max\_year: maximum year to filter data

Returns a pandas dataframe of the selected FeederWatch bird data

clean\_fw\_data: This function cleans the FeederWatch data so that it only contains relevant fields, accurate (valid) data, specified birds, and specified locations.

Args:

* data: a pandas dataframe; raw data downloaded from FeederWatch site
* birds: species data - output of get\_species\_codes \* Note: It can be a subset of this data (e.g., a specific family)
* sub\_national\_code: list of subnational\_code fields to filter to

Returns a subset of the original data input, with cleaned field names.

batchBirdAnalysis: Batch processing and analysis of FeederWatch bird data.

Steps:

1. Creates Feature Class from .csv file in file Geodatabase
2. Adds projection, saving to new Feature Class
3. Filters Projected Feature Class by species, saving individual species to their own Feature Classes in the database

Args:

* fw\_file: FeederWatch data .csv file
* base\_fc: Base Feature Class name
* existing\_fcs: List of existing Feature Classes already saved to the database (if they already exist, they will be skipped during batch processing)
* out\_coordinate\_system: Projected coordinate system
* data\_path: Path to FeederWatch data
* fw\_df: FeederWatch dataframe
* species\_df: Species dataframe
  1. **Class:**Below are two classes used in the project, along with their respective docstrings. One is a subclass of the other, so it makes sense to include them both here.

Species: A class to organize raw FeederWatch dataframes into human-understandable metadata.

Attributes:

* species\_code : (list) List of species codes.
* species\_name : (list) List of species names.
* family : (list) List of families for each species.

Parameters:

* dataframe : (pandas.DataFrame) Raw FeederWatch dataframe containing information on bird species.

Bird: A class to create a bird object from a FeederWatch dataframe. *It is a subclass of Species.*

Attributes:

* code : (str) Bird species code.
* name : (str) Bird species name.
* family : (str) Bird family.
* formatted\_name : (str) Bird name formatted for use as feature class name attribute.
* fc\_name : (str) Feature class name attribute for the bird.

Parameters:

* dataframe : (pandas.DataFrame) Raw FeederWatch dataframe containing information on bird species.
* bird\_name : (str) Name of the bird species to create.
* \_prefix : (str) *optional*;Prefix to be added to the feature class name attribute, by default "FW\_".
  1. **Mapping:**  The model outputs for each of the top MaxEnt Models will be added to individual maps (one for each species).
  2. **GUI:**

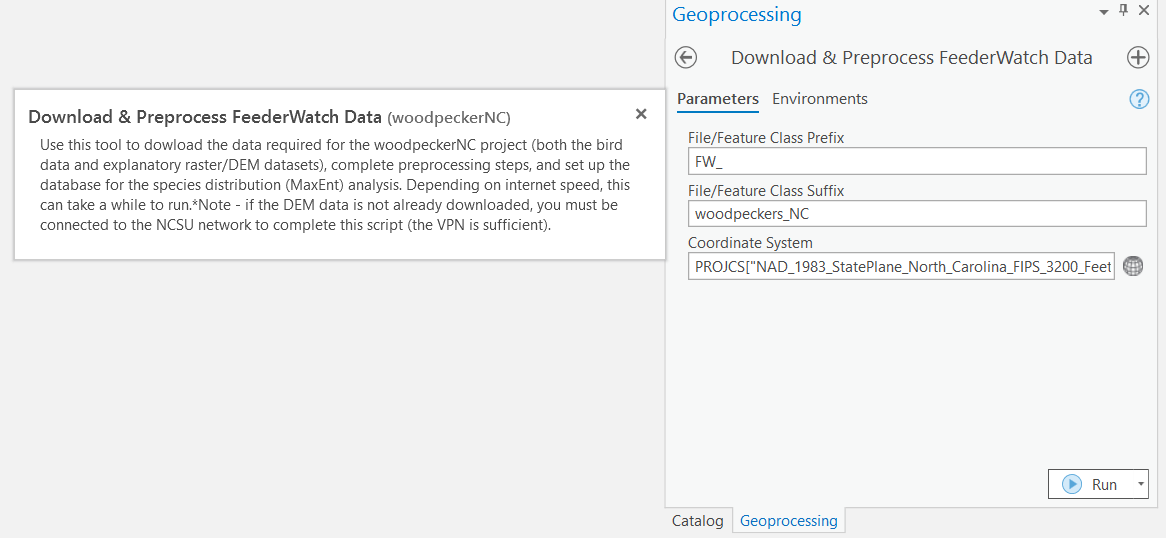


Figure 2: Geoprocessing Tool GUI Screenshot

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Parameter** | **Data Type** | **Type** | **Direction** | **Default Value** | **Filter** |
| \_prefix | String | Optional | Input | FW\_ |  |
| \_suffix | String | Optional | Input | woodpeckers\_NC |  |
| coord\_sys | Coordinate System | Required | Input | NAD\_1983\_StatePlane\_North\_Carolina\_FIPS\_3200\_Feet |  |

*Table 1 : Geoprocessing Tool Parameters*

**Category II**

The item in Category II that will for certain be included in this project is (b) *Downloading and uncompressing data from the web*. As discussed throughout this report, a big part of the project is the retrieval/processing. Data is programmatically downloaded from the web from several sources (in several formats).

**Category III**

At the very least, I will be utilizing item (m): *Inform the user of progress, using messages*. Depending on time, I might try to include one or two more of the suggested items in Category III. However, I feel that this one is essential, especially with how long some of the script can take to run. Keeping a log is helpful when tracking progress and debugging.

**References**

1. <https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/presence-only-prediction.htm>
2. <https://feederwatch.org/about/project-overview/>
3. <https://www.lib.ncsu.edu/gis/nlcd>
4. <https://www.mrlc.gov/>
5. <https://www.lib.ncsu.edu/gis/dem>
6. <https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/presence-only-prediction.htm>