**Evaluating environmental predictors of breeding waterfowl population distribution and abundance in the Central Interior Plateau of British Columbia (2007-2017)**

*Yuriko Hashimoto*

Canadian Wildlife Service, Pennsylvannia State University

ABSTRACT

The Waterfowl Breeding Population and Habitat Survey has been described as “arguably the largest and best-designed population survey in the world” (Murray, Anderson, & Steury, 2010)

Keywords: waterfowl, machine learning, random forest, species distribution model

|  |
| --- |
| image  Figure 1. The survey area divided into ecosections of the Ecoregions of British Columbia (Demarchi, 2011). Latitudinal strip transects are spaced ten miles (16 km) apart within the British Columbia’s Central and Sub-boreal plateaus. |

INTRODUCTION

While the uses and practical applications of species distribution models in conservation decision-making are diverse (Guillera-Arroita et al., 2015; Guisan & Thuiller, 2005; Guisan et al., 2013)—the breadth of which includes predicting climate change impacts and biological invasions, critical habitat identification, reserve selection and protected areas management and translocation of species and habitats (Elith & Graham, 2009) just to name a few—the overarching aim is to conserve biodiversity and support better land use planning and environmentally sustainable management practices.

Recognizing the lack of waterfowl species distribution models (SDMs) to support conservation actions in response to the urgency of conservation challenges in Canada’s boreal forests, Barker et al (2014) published the first national-scale waterfowl SDMs in 2014 based on the traditional Waterfowl Breeding Population and Habitat Survey (WBPHS) database. The WBPHS has been described as “arguably the largest and best-designed population survey in the world” (Murray et al., 2010), continental in scale and running on a continuous annual basis since 1955, the survey captures what is recognized to be “core waterfowl breeding habitat” in North America. Designed primarily to provide annual breeding population estimates and inform hunting regulations in the U.S. and Canada, these data additionally provide a long-term population monitoring dataset that has since informed countless studies on species-habitat relationships in support of waterfowl conservation. While the models performed well over all, the results of extrapolation to out-of-sample areas were variable for a number of reasons as outlined by the authors. British Columbia is not within the traditional survey and no survey data from BC informed the models which predicted low total densities for the Western Cordillera between the Pacific coast and Rocky Mountains—a result that conflicts with anecdotal expert opinion (A. Breault, personal communication 2018). As such there are currently no adopted waterfowl breeding population models in use for British Columbia. This study aims to fill the this gap and help support the mandate of the Canadian Wildlife Service (CWS) to conserve biodiversity (Environment and Climate Change Canada, 2019; Environment Canada - Biodiversity Convention Office, 1995).

In the early 2000s after exploratory pilot surveys in BC’s Central and Sub-boreal highlands determined waterfowl population abundances to be significant enough to justify a regional breeding survey program, the British Columbia Breeding Waterfowl Survey began in earnest in 2006. BC’s May surveys, run jointly by the CWS and the U.S. Fish and Wildlife Service and conducted in partnership with Ducks Unlimited Canada, inform the annual population status of migratory game birds in the Central Interior Plateau of BC and contributes to adaptive harvest strategies for mallards in the Pacific Flyway (Zimpfer, Breault, & Sanders, 2019).

After consolidating and standardizing the annual survey data, I developed methods to create SDMs for the top ten most abundant species as well as five groups based on nesting and feeding guilds (Baldassarre, 2014) (Table #). Guisan et al (2013) have observed that despite the oft-cited assumption of the utility of SDMs in conservation decision-making there is little evidence demonstrating their application in real-world conservation management. Following their recommendation that modellers make explicit the objectives of the framework within which models were developed, I provide guidance on how the models may be improved and implemented as a decision-making platform for conservation planning by the Canadian Wildlife Service and partners in conservation.

MATERIALS AND METHODS

Survey Methods

The study area was designed to capture BC’s prime waterfowl breeding habitat of the humid, continental plateaus of the Central and Sub-boreal Interior between the Coast mountains to the west and Rocky Mountains to the east. The region of interlocking highlands and valleys are mainly forested and sparsely populated. The main industries include forestry, agriculture and mining (Demarchi, 2011). The survey design consists of latitudinal strip transects, 400 metres wide, spaced ten miles apart (16.09 km) and delineated by the boundaries of eight ecosections which represent the finest scale, sub-regional stratum within the Ecoregion Classification System of British Columbia—a small-scale ecosystem management framework stratifying regions of “similar climate, physiography, oceanography, hydrology, vegetation and wildlife potential” (Demarchi, 2011) (Figure 1). The surveys are conducted by helicopter and surveyed by a consistent survey crew of two to three experienced observers (Zimpfer et al., 2019). The transects when originally conceived were designed within the NAD 1983 geographic coordinate system and based on the mapped Ecoregion Classification System version 2.0 (1995). The annual surveys continue to follow the extent of the original design however the ecosection boundaries have since been updated with finer scale vegetation zonation data in version 2.1 (2006) and all analyses were based on the areal overlap between the two versions.

The survey does not follow the design methodology of the traditional WBPHS, however the survey techniques are consistent with the methods outlined in Smith (1995) and the Standard Operating Procedures (US Fish and Wildlife Service (USFWS) and Canadian Wildlife Service (CWS), 1987). Prior to 2010, utilized survey technologies consisted of paper maps, field notes and GPS units with navigation determined by piloting along bearings to GPS waypoints. In 2010, the mobile GIS software PC-Mapper with Airborne Inspection (version 4.0, Corvallis Microtechnology Inc, 2015) was adopted for both survey navigation and data collection. The software is run on Panasonic Toughbooks (CF-19 and CF-31) with the screen in view of both the pilot and observer. Real-time navigation is guided by the GIS with base data that includes strip transect boundaries, freshwater polygons and stream segments, ecosection boundaries and fuel waypoints. Digital data collection via georeferenced voice recordings transcribed by the observer post-survey collection have replaced paper analogue methods.

The survey is designed to capture the primary breeding period beginning in early May but is weather and climate dependent and has taken place as early as late April (April 28, 2015) and as late as mid-May (May 13, 2011) during a particularly warm and cold spring, respectively. The study area extent is almost 11 million hectares and takes an average 23 days (~105 flying hours) to survey.

**Population estimates**

The surveys are conducted by helicopter along meandering paths at altitudes and speeds lower than fixed-wing aircraft (30-50 m and 40-80 km/h, respectively) therefore no complementary ground surveys are conducted and no visibility correction factor is applied as visibility is assumed to be complete. The total indicated breeding population estimates were derived from raw counts based on species, sex and grouping as outlined in Smith (1995). The temporal subset of the analysis was from 2007 to 2017 inclusive (data from 2006 was excluded due to data inconsistencies and 2018 was excluded due to the absence of interpreted climate data). Survey observation points record the location of the observer within the helicopter and not the bird on the ground. Efforts to correct locations guided by reference data and field observation notes were discontinued from 2012 onward due to resource constraints and inconsistent observational data. In order to account for this spatial uncertainty, observation location points were collated to the nearest 400m interval along the center line of the strip transect.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C:\Users\hashimotoy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6711D398.tmp   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | C:\Users\hashimotoy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6711D398.tmp | Babine Upland | C:\Users\hashimotoy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6711D398.tmp | Cariboo Basin | C:\Users\hashimotoy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6711D398.tmp | Nazko Upland | C:\Users\hashimotoy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6711D398.tmp | Quesnel Lowland | | Bulkley Basin | Chilcotin Plateau | Nechako Lowland | Western Chilotin Upland | |
| Figure 2. Total indicated breeding population of the top ten most common species within each ecosection (2007-2017). |

**Environmental** **data**

Environmental predictor variables were pre-selected based on ecological theory of life history traits and physiological processes, hydrological features associated with wetlands, and land use practices and disturbances that can potentially influence waterfowl habitat. Where possible direct measures of predictor variables rather than proxy data was selected. For example, measures of temperature and precipitation rather than the indirect measure of elevation were included in the analyses. A number of datasets were collated and evaluated but the final data inputs were constrained by availability, resolution, coverage, and currency, and complicated by interpretation and relevance (Appendix 1). A small sample of distributions is presented (Figure #).

Annual, seasonal, monthly and 30-year normal climatic variables were extracted from ClimateBC software (version 6.10, Wang, Hamann, Spittlehouse, & Carroll, 2016) based on the coordinates and elevation at 400m interval points along the transect center line. Growing degree days and average April temperature were included to capture primary productivity. To represent and characterize hydrological regimes which are driven mainly by snowpack accumation in winter within the study area (Demarchi, 2011; Islam et al., 2017; R. Pike, Redding, Moore, Winkler, & Bladon, 2010) precipitation as snow and the climatic moisture deficit (precipitation minus potential evapotranspiration) were selected. The Biogeoclimatic Zones of British Columbia (BEC) is a standardized provincial dataset that classifies ecosystems within nested classes of regional, site and chronological levels of apex vegetaion (Ministry of Forests, Lands, Natural Resource Operations and Rural Development (FLNRO), 2018) and was included to represent the combined influence of soil chemistry, vegetation, topography and climate. Additionally, the BEC Zone classifications provide an alternative regional classification system to the Ecoregions of BC ecosections for supplemental model development.

Land cover variables to characterize habitat were derived from 2010 Landsat imagery published by the North American Land Change Monitoring Sytem (CCRS/CCMEO/NRCan, 2017). Cover classes were aggregated and reclassified to account for imbalanced classes (Appendix #). Topographic data included slope and aspect derived from 1:20,000 DEM (FLNRO, 2014).

Hydrological variables were derived from the provincial reference dataset for standardized hydrological features, and included polygons of lakes, rivers, man-made waterbodies and wetlands as well as stream lines (FLRNO, 2011). Lakes were classified by size class and streams were aggregated by stream order values (Appendix #). To account for shoreline complexity and avoid correlation between lake area and perimeter, a lake perimeter:area index was derived.

The line network of unpaved roads was included to represent anthropogenic disturbance of resource extraction activities (Ministry of Forests, Lands, Natural Resource Operations and Rural Development, 2013).

Land management practices were represented by the Agricultural Land Reserve (Agricultural Land Commission, 2018)—a provincial designation designed to identify and conserve agricultural productive lands, and by the Protected Areas Database of lands under federal, provincial and municipal protection and land conservancy trusts (CWS, 2018).

**Geoprocessing Methods**

Spatial analyses were performed and mapping products were produced in ArcGIS Pro (versions 2.2 to 2.4, ESRI, 2019) and Python (version 3.6.5, Python Software Foundation, 2018). Predictor variables were extracted, projected, rasterized and generalized as required in BC Albers equal area coordinate system within a 1 km buffer of the study area to eliminate edge effects. The location uncertainty of point observations determined the finest scale of the analysis was a resolution of 400m. Topographic values of slope and aspect were generalized to the mean average within 16 ha (400m x 400m) cells determined by the boundaries of the strip transect and 400m interval breaks. BEC zone classifications were generalized by majority area within the cell and remained categorical. All remaining predictors were represented by continuous values and generalized to a 1.2 km radius of the interval centroid to capture landscape level effects (Figure #). Predicted response values were projected to a fishnet grid of 400m cell centroid points with attributed predictor values and rasterized for output and display. Many of the steps required in the pre-processing of spatial data and map production were automated in a Python Script Toolbox (Appendix #).

|  |
| --- |
| **C:\Users\hashimotoy\Documents\MGIS\Documentation\Reports\Jupyter\zoom_landscape.png**  Figure 3. Representative schematic of the survey methods incorporating survey data from 2017. Observation points (yellow dots) are collected along the meandering flight tracklog (dashed pink line) within the transect. The strip transect was spliced into 400m wide intervals (red squares) with a circular focal neighbourhood of 1.2 km (pale circle) of the 400x400m grid cell centroid (black dot) used to characterize landscape level effects of environmental variables. |

**Statistical Methods**

Random forest is an ensemble machine learning algorithm that creates a series of decision trees based on the principles of bagging (or bootstrapped aggregation—the drawing of a large number of data samples by random sampling with replacement ) and random permutation. Each individual decision tree is based on a random subset of the data and at each node of the tree the data is partitioned into two branches based on a randomly selected but predetermined number of predictor variables evaluated by the algorithm to produce the best split. As individual trees are highly susceptible to noise in the data and sensitive to local optima, each tree is considered a ‘weak learner’ (Guisan, Wilfried, & Zimmermann, 2017). But by combining and averaging the results of several regression trees, a ‘strong learner’ is created in the final ensembled prediction.

The random forest approach for building the SDMs was chosen for its predictive accuracy, resistance to overfitting, ability to account for imbalanced classes, ability to handle both continuous and categorical variables, and its independence from requirements of feature scaling and centering as well as assumptions of normality (Cutler, Cutler, & Stevens, 2012; Guisan et al., 2017). The observation frequency distribution of bird counts reflected a zero-inflated negative binomial distribution typical of ecological count data (Qian, 2010) but not amenable to standard regression-based approaches.

All statistical modeling and data manipulation was performed in R (version 3.5.3, R Core Team, 2018). The R ‘party’ package (version 1.3-3) was selected for its implementation of the random forest and bagging algorithm ‘cforest\_unbiased’ function which utilizes conditional inference trees that account for correlation structures between variables in the permutation scheme of variable selection; additionally, the function implements a permutation importance measure, function ‘varimp’, that is immune to erroneous calculations due to correlated responses and is not biased towards continuous data or categorical variables with many classifications unlike the Gini accuracy of traditional random forest implementations (Strobl, Boulesteix, Kneib, Augustin, & Zeileis, 2008; Strobl, Hothorn, & Zeileis, 2009). Separate models for each species or species group within each ecosection were developed due to regional differences in predictor variables and for computational efficiency. Slight differences in selected predictor variables between ecosection models were due to zero or near-zero variance, correlated data structures and/or mismatched factor variables in ecosection-based data inputs for model training and prediction.

Preliminary results of model inputs of annual survey data resulted in low variable importance rankings for the dynamic climate variables. As all other environmental data was static the model input values were based on mean averaged counts and climate measures. Due to high zero-inflation (average of 93.6% frequency of zeroes for the top ten most common species) the data was not subset into training, validation, and test sets to evaluate model performance, instead the internal out-of-bag (OOB) error measures were determined. As each tree is based on a random subset of the training data, a collection of datasets which do not contain a particular record can be ensembled for each record. This collection forms the out-of-bag examples which are used as an unbiased test set to assess prediction accuracy by averaging the error rate. The predetermined number of candidate variables for each node split (‘mtry’ parameter of ‘cforest’) was left at the default value of 5 which resulted in better OOB estimates than Briemann’s rule of thumb to divide the number of candidate variables by three for regression trees (Guisan et al., 2017).

The conditional inference trees utilized by ‘cforest’ can tolerate correlated data structures, but to reduce processing time, data preparation included addressing correlation and multicollinearity by step-wise elimination of correlated variables with threshold measures of r = 0.8 followed by elimination of variance inflation factors of 5 and greater as recommended by Guisan et al (2017).

Random forest methods are widely recognized as fast and efficiently able to handle large amounts of data and model variables but the unbiased algorithm of the ‘cforest’ implementation is more computationally intensive than standard approaches. For example model training took approximately 10 minutes to process ~3,000 records and 27 variable inputs for mallards in Babine Upland, while model forecasting to the ecosection, an area ~ 40 times greater, took over 18 hours on a 64-bit OS workstation with 48.0 GB RAM and an Intel Xeon(R) CPU 3.60GHz.

The R scripts developed for statistical analyses as well as links to model data inputs and an interactive Jupyter Notebook are provided in Appendix #.

RESULTS AND DISCUSSION

Model Performance

The training models and predicted outputs are representations of the complex ecological interactions and associations between the predictor variables and the species or species group under study. Predictive modelling algorithms like random forest forsake the theoretical hypotheses of explanatory causal models for accurate forecasting (Shmueli, 2011).

Model performance was assessed with the out-of-bag (OOB) error. Table # lists the variable importance measures and relative rank.

Variable importance measures may reveal causal mechanisms to inform the development of explanatory models (Shmueli, 2011).

Species Abundance and Distribution

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **C:\Users\hashimotoy\Documents\MGIS\Documentation\Reports\Jupyter\rel_pop_sp_eco.png**   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | C:\Users\hashimotoy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6711D398.tmp | Babine Upland | C:\Users\hashimotoy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6711D398.tmp | Cariboo Basin | C:\Users\hashimotoy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6711D398.tmp | Nazko Upland | C:\Users\hashimotoy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6711D398.tmp | Quesnel Lowland | | Bulkley Basin | Chilcotin Plateau | Nechako Lowland | Western Chilotin Upland | |
| Figure 4. Boxplots of relative annual population density per square kilometre of the top ten most common species by ecosection. |
| **C:\Users\hashimotoy\Documents\MGIS\Documentation\Reports\Jupyter\rel_pop_sp_eco.png**   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | C:\Users\hashimotoy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6711D398.tmp | Babine Upland | C:\Users\hashimotoy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6711D398.tmp | Cariboo Basin | C:\Users\hashimotoy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6711D398.tmp | Nazko Upland | C:\Users\hashimotoy\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6711D398.tmp | Quesnel Lowland | | Bulkley Basin | Chilcotin Plateau | Nechako Lowland | Western Chilotin Upland | |
| Figure 4. Boxplots of relative annual population density per square kilometre of the top ten most common species by ecosection. |

Model Limitations

We know that “[d]ucks like water” (Pimm, 1994), but where this water is we do not actually know: Canada, unlike most industrialized nations, does not have a national wetland dataset (Canadian Wetland Inventory — Ducks Unlimited Canada, 2019). The delineation of highly productive wetlands, including ephemeral wetlands due to increased nutrient mineralization by aerobic microbes (Schlesinger & Bernhardt, 2013) and smaller wetlands due to increased light penetration at shallower depths and increased shoreline vegetation, are difficult to capture with freely available, coarser-scale remote sensing imagery (e.g. Landsat imagery at 30m resolution). Techniques for finer-scale wetland feature extraction based on Landsat have been developed for open wetlands but have not yet been fully developed and tested on forested wetlands (Halabisky et al., 2018). Relative to other provinces, BC has an abundance of fine-scale (1:20,000) base reference spatial data however the creation of much of these data has been driven by the forestry sector which is a significant component of the BC economy. Consequently, much of the mapping of environmental features has been focused on supporting forestry management practices and the delineation of wetlands (areas unsuited to logging) especially at higher elevations is poor (D. Filatow, personal communication April 28, 2018). Ongoing efforts to develop methodologies for wetland mapping coincident with the May surveys utilizing Radarsat-2 technologies in a collaborative effort between the CWS and the Science and Technology branches of Environment and Climate Change Canada are promising but results are still preliminary and the project is far from complete (K. Moore, personal communication, September 30, 2018). Additionally, the Canadian Wetland Inventory project spearheaded by Ducks Unlimited Canada aims to provide a comprehensive national wetland inventory but BC has yet to be mapped (Canadian Wetland Inventory — Ducks Unlimited Canada, 2019). Model performance and predictive accuracy can be expected to improve with the incorporation of improved wetland cover products, but proactive management measures should not be delayed while waiting for these data.

Hydrological regimes that create and maintain wetland ecosystems are influenced by a host of factors such as the depth of snowpack, the rate of snowmelt, levels of groundwater, precipitation, glacial retreat, land management practices, and anthropogenic and natural disturbance (Adamus, 2014; R. G. Pike et al., 2010). Future modeling exercises may be improved by incorporating snow basin, drainage and/or watershed data (as identified in Appendix #). BC’s forested ecosystems have been challenged by the mountain pine beetle outbreak from 1999 to 2015, followed by the extreme fire seasons of 2017 and 2018, the impacts of which on forest hydrology are difficult to predict (R. G. Pike et al., 2010; R. Pike et al., 2010) yet worthy of future exploration (Bunnell, Fred L, Wells, Ralph, 2010; MacKenzie & Moran, 2004; Snauffer, Hsieh, & Cannon, 2016).

The population estimates reflect sampled observations over a limited period of time and therefore reflect only a snapshot of the species-habitat relationship assumed to be in pseudo-equilibrium (Guisan et al., 2017). Moreover, the models did not account for density-dependence, breeding philopatry, site fidelity, territoriality, influence of breeding phenology, species nesting chronologies or lagged response to environmental shifts. Recent studies in breeding phenology of cavity-nesting birds identified critical temperature periods of local temperature as few as 4 days (Drake & Martin, 2018). Daily temperature datasets from Natural Resources Canada (McKenney et al., 2011) at 1 km resolution was collated but not included in the analyses. Datasets such as this that may improve future model development are identified in Appendix #.

Model Applications

This modeling study had two main, related objectives: first, to develop the methods and techniques required to standardize and consolidate the BC May Surveys for analysis and secondly, to create SDMs of relative abundance and distribution in support of breeding waterfowl conservation and habitat management. The results of the models can be used to highlight hotspot areas of high abundance and diversity for evaluation of protection or mitigation measures, provide finer-scale population estimates for environmental impact assessments and inform the calculation of offsets, support adaptive management measures with respect to climate change, provide a basis for further studies of ecological relationships, and serve as guidance for future modeling exercises to account for model limitations highlighted above.

Changes in hydrological regimes due to increased warming and drying trends are predicted to lead to wetland losses at low to mid-elevations however higher elevations are less sensitive to temperature changes that can affect snowpack accumulation (Bunnell, Fred L, Wells, Ralph, 2010; R. G. Pike et al., 2010; R. Pike et al., 2010). While BC’s central interior waterfowl breeding habitat is not as productive as the Prairie Pothole Region (PPR), the region’s conservation value, especially at elevations greater than 1200 metres (Bunnell, Fred L, Wells, Ralph, 2010), may rise in importance with the documented and predicted climate-change related shifting of wetland productivity in the PPR (N. D. Niemuth, Fleming, & Reynolds, 2014; N. Niemuth, Wangler, & Reynolds, 2010; Zhao, Silverman, Fleming, & Boomer, 2016). Future studies of climate impacts should take into account the wide ranges in variability of projected climate between the global models available in ClimateBC and employ ensemble methods that encompass this range of variability for impact assessments and adaptive management (R. G. Pike et al., 2010; Spittlehouse & Wang, 2016).

CONCLUSION

“The single story creates stereotypes, and the problem with stereotypes is not that they are untrue,

but that they are incomplete. They make one story become the only story“ (Adichie, 2009).

With the quote above, the Nigerian author Chimamanda Ngozi Adichie was referring to the dangers of racial stereotyping. But similarly, a species distribution model is a single story: a simplified human construct of a complex ecological phenomenon shared to help reveal patterns, better our understanding and aid us in assessing our conservation efforts and in planning conservation action. Models, like stories, must be developed and communicated in meaningful ways. Future modeling efforts and applications are encouraged to validate the models with emerging data, integrate best available methods and data, address and account for uncertainty, and to support continued systematic monitoring and ecological research to better our understanding of fundamental biological and ecological theory (Sinclair et al, 2010). This study is only one story but it is a story worth telling in support of waterfowl conservation in British Columbia.

ACKNOWLEDGEMENTS

Throughout this project I have been provided with a great deal of support and assistance. Foremost, I sincerely thank Dr. Julian Avery, Beth King, and Dr. Justine Blanford at Penn State for their invaluable guidance and unflagging support. I express gratitude to the Canadian Wildlife Service and Fisheries and Oceans Canada who as my employers provided the time and resources to carry out and complete the study. My CWS colleagues—Andre Breault, Kathleen Moore, Dr. Rhonda Millikin, Dr. Andrea Norris, Jeffrey Thomas and Dr. Caroline Fox—who provided indispensable guidance and encouragement and generously shared their time and wealth of knowledge. Finally, I thank my partner Brady Ciel Marks for her enduring patience, humour, and awesome dance moves.

REFERENCES

Adamus, P. (2014). Effects of Forest Roads and Tree Removal In or Near Wetlands of the Pacific Northwest: A Literature Synthesis, (December).

Adichie, C. N. (2009). The Danger of a Single Story. Ted Global. Retrieved from https://www.ted.com/talks/chimamanda\_adichie\_the\_danger\_of\_a\_single\_story?language=en

Baldassarre, G. (2014). *Ducks, Geese, and Swans of North America* (Revised an). John Hopkins University Press.

Bunnell, Fred L, Wells, Ralph, M. A. (2010). Vulnerability of wetlands to climate change in the Southern Interior Ecoprovince : a preliminary assessment 1 Final Report. *Centre for Applied Conservation Research, University of British Columbia*, (March).

Canadian Wetland Inventory — Ducks Unlimited Canada. (n.d.). Retrieved July 27, 2019, from https://www.ducks.ca/initiatives/canadian-wetland-inventory/

CCRS/CCMEO/NRCan. (2017). 2010 Land Cover of North America at 30 meters. Retrieved from http://www.cec.org/tools-and-resources/map-files/land-cover-2010-landsat-30m

Commission, A. L. (2018). Agricultural Land Reserve. Agicultural Land Commission. Retrieved from https://catalogue.data.gov.bc.ca/dataset/alc-alr-boundary

Corvallis Microtechnology Inc. (2015). PC-Mapper with Airborne Inspection. Corvallis, Oregon.

Cutler, A., Cutler, D., & Stevens, J. (2012). Random Forests. In C. Zhang & Y. Ma (Eds.), *Ensemble Machine Learning*.

Demarchi, D. A. (2011). *An Introduction to the Ecoregions of British Columbia*. Victoria. Retrieved from http://www.env.gov.bc.ca/wld/documents/techpub/rn324.pdf

Drake, A., & Martin, K. (2018). Local temperatures predict breeding phenology but do not result in breeding synchrony among a community of resident cavity-nesting birds. *Scientific Reports*, *8*(1), 2756. https://doi.org/10.1038/s41598-018-20977-y

Elith, J., & Graham, C. H. (2009). Do they? How do they? WHY do they differ? On finding reasons for differing performances of species distribution models. *Ecography*, *32*(1), 66–77. https://doi.org/10.1111/j.1600-0587.2008.05505.x

Environment and Climate Change Canada. (2019). *Summary of Canada’s 6th National Report to the Convention on Biological Diversity*. Gatineau, Quebec.

Environment Canada - Biodiversity Convention Office. (1995). *Canadian Biodiversity Strategy--Canada’s Response to the Convention on Biological Diversity*. Hull, Quebec. Retrieved from http://www.biodivcanada.ca/560ED58E-0A7A-43D8-8754-C7DD12761EFA/CBS\_e.pdf

ESRI. (2019). ArcGIS Pro. Redlands.

Guillera-Arroita, G., Lahoz-Monfort, J. J., Elith, J., Gordon, A., Kujala, H., Lentini, P. E., … Wintle, B. A. (2015). Is my species distribution model fit for purpose? Matching data and models to applications. *Global Ecology and Biogeography*, *24*(3), 276–292. https://doi.org/10.1111/geb.12268

Guisan, A., & Thuiller, W. (2005). Predicting species distribution: offering more than simple habitat models. *Ecology Letters*, *8*(9), 993–1009. https://doi.org/10.1111/j.1461-0248.2005.00792.x

Guisan, A., Tingley, R., Baumgartner, J. B., Naujokaitis-Lewis, I., Sutcliffe, P. R., Tulloch, A. I. T., … Buckley, Y. M. (2013). Predicting species distributions for conservation decisions. *Ecology Letters*, *16*(12), 1424–1435. https://doi.org/10.1111/ele.12189

Guisan, A., Wilfried, T., & Zimmermann, N. E. (2017). *Habitat Suitability and Distribution Models with Applications in R*. Cambridge University Press. https://doi.org/10.1017/9781139028271

Halabisky, M., Babcock, C., Moskal, L., Halabisky, M., Babcock, C., & Moskal, L. M. (2018). Harnessing the Temporal Dimension to Improve Object-Based Image Analysis Classification of Wetlands. *Remote Sensing*, *10*(9), 1467. https://doi.org/10.3390/rs10091467

Islam, S. ul, Déry, S. J., Werner, A. T., Islam, S. ul, Déry, S. J., & Werner, A. T. (2017). Future Climate Change Impacts on Snow and Water Resources of the Fraser River Basin, British Columbia. *Journal of Hydrometeorology*, *18*(2), 473–496. https://doi.org/10.1175/JHM-D-16-0012.1

MacKenzie, W. H., & Moran, J. R. (2004). *Wetlands of British Columbia: a guide to identification. Land Management Handbook 52*. https://doi.org/Land Management Handbook No. 52

McKenney, D. W., Hutchinson, M. F., Papadopol, P., Lawrence, K., Pedlar, J., Campbell, K., … Owen, T. (2011). Customized Spatial Clinate Models for North America. *Bulletin of the American Meteorological Society*, (December), 1611–1622. https://doi.org/10.1175/BAMS-D-10-3132.1

Ministry of Forests, Lands, Natural Resource Operations, and R. D. (Organization/Institution). (2011). Freshwater Atlas. Government of British Columbia. Retrieved from ftp://ftp.geobc.gov.bc.ca/sections/outgoing/bmgs/FWA\_Public

Ministry of Forests, Lands, Natural Resource Operations, and R. D. (Organization/Institution). (2018). Biogeoclimatic Zones of British Columbia. Victoria: Government of British Columbia. Retrieved from https://catalogue.data.gov.bc.ca/dataset/bec-map

Ministry of Forests, Lands, N. R. O. and R. D. (Organization/Institution). (2013). Digital Road Atlas - Master Partially Attributed Roads. Victoria: Government of British Columbia. Retrieved from https://catalogue.data.gov.bc.ca/dataset/digital-road-atlas-dra-master-partially-attributed-roads/resource/a06a2e11-a0b1-41d4-b857-cb2770e34fb0

Ministry of Forests, Lands, N. R. O. and R. D. (Organization/Institution). (2014). TRIM Contours. Victoria: Government of British Columbia.

Murray, D. L., Anderson, M. G., & Steury, T. D. (2010). Temporal shift in density dependence among North American breeding duck populations. *Ecology*, *91*(2), 571–581. https://doi.org/10.1890/MS08-1032.1

Niemuth, N. D., Fleming, K. K., & Reynolds, R. E. (2014). Waterfowl Conservation in the US Prairie Pothole Region: Confronting the Complexities of Climate Change. *PLoS ONE*, *9*(6), e100034. https://doi.org/10.1371/journal.pone.0100034

Niemuth, N., Wangler, B., & Reynolds, R. (2010). Spatial and temporal variation in wet area of wetlands in the Prairie Pothole Region of North Dakota and South Dakota. *Wetlands*, *30*.

Pike, R. G., Bennett, K. E., Redding, T. E., Werner, A. T., Spittlehouse, D. L., Moore, R. D. D., … Tschaplinski, P. J. (2010). Climate Change Effects on Watershed Processes in British Columbia. In *Compendium of Forest Hydrology and Geomorphology in British Columbia* (Vol. 3). https://doi.org/10.5194/nhess-12-1-2012

Pike, R., Redding, T., Moore, R., Winkler, R., & Bladon, K. (2010). *Compendium of forest hydrology and geomorphology in British Columbia*. Retrieved from https://www.for.gov.bc.ca/hfd/pubs/docs/lmh/Lmh66.htm

Pimm, S. L. (1994). The importance of watching birds from airplanes. *Trends in Ecology & Evolution*, *9*(2), 41–43. https://doi.org/10.1016/0169-5347(94)90264-X

Python Software Foundation. (2018). Python Language Reference. Retrieved from https://www.python.org/downloads/

Qian, S. S. (2010). *Environmental and Ecological Statistics with R*. CRC Press.

R Core Team. (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. Retrieved from http://www.r-project.org/

Schlesinger, W. H., & Bernhardt, E. S. (2013). *Biogeochemistry* (3rd ed.). Elsevier. https://doi.org/https://doi.org/10.1016/C2010-0-66291-2

Shmueli, G. (2011). To Explain or to Predict? *Statistical Science*, *25*(3), 289–310. https://doi.org/10.1214/10-sts330

Sinclair, S., White, M., & Newell, G. (2010). How useful are species distribution models for managing biodiversity under future climates? *Ecology and Society*, *15*(1). Retrieved from http://www.ecologyandsociety.org/vol15/iss1/art8/

Smith, G. (1995). *A Critical Review of the Aerial and Ground Surveys of Breeding Waterfowl in North America*. *Biological Science Report* (Vol. 268). https://doi.org/10.1126/science.268.5215.1262

Snauffer, A. M., Hsieh, W. W., & Cannon, A. J. (2016). Comparison of gridded snow water equivalent products with in situ measurements in British Columbia, Canada. *Journal of Hydrology*, *541*, 714–726. https://doi.org/10.1016/J.JHYDROL.2016.07.027

Spittlehouse, D., & Wang, T. (2016). Comparison of Climate Change Projections in ClimateBC v5 . 30, 1–6.

Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T., & Zeileis, A. (2008). Conditional variable importance for random forests. *BMC Bioinformatics*, *9*(1), 307. https://doi.org/10.1186/1471-2105-9-307

Strobl, C., Hothorn, T., & Zeileis, A. (2009). Party on! A new, conditional variable-importance measure for random forests available in the party package. *R Journal*, *1*(2), 14–17.

US Fish and Wildlife Service (USFWS) and Canadian Wildlife Service (CWS). (1987). *Standard operating procedures for aerial waterfowl breeding ground population and habitat surveys in North America*.

Wang, T., Hamann, A., Spittlehouse, D., & Carroll, C. (2016). Locally Downscaled and Spatially Customizable Climate Data for Historical and Future Periods for North America, 1–17. https://doi.org/10.1371/journal.pone.0156720

Zhao, Q., Silverman, E., Fleming, K., & Boomer, G. S. (2016). Forecasting waterfowl population dynamics under climate change — Does the spatial variation of density dependence and environmental effects matter? *Biological Conservation*, *194*, 80–88. https://doi.org/10.1016/J.BIOCON.2015.12.006

Zimpfer, N. L., Breault, A., & Sanders, T. (2019). *British Colulmbia Breeding Waterfowl Survey Report, 2019*.

APPENDIX # – Dataset Evaluation

Environmental datasets collated and evaluated for the modelling exercise. The table describes each dataset and provides a brief description of variables and their assessment.

APPENDIX

APPENDIX 2 – Reclassification

APPENDIX 3 – Python Script Toolbox

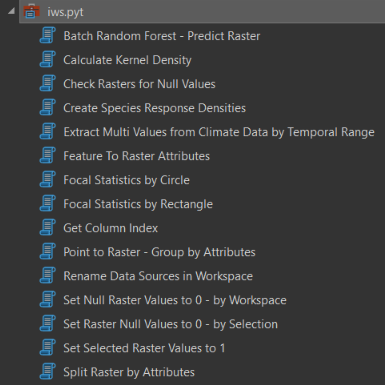


Figure 5. Python Script Toolbox as displayed in ArcGis Pro.

The code for the Python script toolbox is provided below.

# -\*- coding: utf-8 -\*-  
  
import arcpy, os  
from arcpy import env  
arcpy.env.overwriteOutput = True  
from arcpy.sa import \*  
arcpy.CheckOutExtension("Spatial")  
arcpy.env.qualifiedFieldNames = False  
# scratch\_ws = arcpy.CreateScratchName(workspace=arcpy.env.scratchGDB)  
arcpy.env.snapRaster = r"C:\Users\hashimotoy\Desktop\ws\base\_alb.gdb\dem"  
arcpy.env.mask = r"C:\Users\hashimotoy\Desktop\ws\base\_alb.gdb\survey\_mask"  
scratch\_ws = r"C:\Users\hashimotoy\Desktop\ws\_prep\scratch.gdb"  
  
def checkFieldExists(in\_tbl, field\_nm, field\_type):  
 fields = arcpy.ListFields(in\_tbl)  
 x = False  
 for field in fields:  
 if field.name == field\_nm:  
 x = True  
 if x == False:  
 arcpy.AddField\_management(in\_tbl, field\_nm, field\_type)  
  
  
def getBaseName(in\_fc):  
 desc = arcpy.Describe(in\_fc)  
 basename = desc.baseName  
 return basename  
  
def unique\_values(table, field):  
 with arcpy.da.SearchCursor(table, [field]) as cursor:  
 return sorted({row[0] for row in cursor})  
  
  
class Toolbox(object):  
 def \_\_init\_\_(self):  
 *"""Define the toolbox (the name of the toolbox is the name of the  
 .pyt file)."""* self.label = "IWS Data Tools"  
 self.alias = ""  
  
 # List of tool classes associated with this toolbox  
 self.tools = [featureToRasterAttributes,  
 setRasterNullValuesTo0,  
 setRasterNullValuesTo0Workspace,  
 renameDataInWorkspace,  
 kernelDensityCalculations,  
 splitRasterByAttributes,  
 focalStatsByCircle,  
 focalStatsByRectangle,  
 setRasterValuesTo1,  
 pointToRasterGroupByAttributes,  
 getColumnIndex,  
 extractClimateValuesForRange,  
 checkRasterForNullValues,  
 batchRandomForest,  
 createPredictedDensitySurfaces]  
  
class Tool(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Tool"  
 self.description = ""  
 self.canRunInBackground = False  
  
 def getParameterInfo(self):  
 *"""Define parameter definitions"""* # First parameter  
 param0 = arcpy.Parameter(  
 displayName="Input Features",  
 name="in\_features",  
 datatype="GPFeatureLayer",  
 parameterType="Required",  
 direction="Input")  
  
 # Second parameter  
 param1 = arcpy.Parameter(  
 displayName="Sinuosity Field",  
 name="sinuosity\_field",  
 datatype="Field",  
 parameterType="Optional",  
 direction="Input")  
  
 param1.value = "sinuosity"  
  
 # Third parameter  
 param2 = arcpy.Parameter(  
 displayName="Output Features",  
 name="out\_features",  
 datatype="GPFeatureLayer",  
 parameterType="Derived",  
 direction="Output")  
  
 param2.parameterDependencies = [param0.name]  
 param2.schema.clone = True  
  
 parameters = [param0, param1, param2]  
  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
 *"""The source code of the tool."""* return  
  
class batchRandomForest(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Batch Random Forest - Predict Raster"  
 self.description = "Tool uses explanatory rasters to predict to raster. Specify the output workspace" \  
 "for the four resulting outputs: prediction raster, variable importance table, trained " \  
 "features and validation r2. The outputs will be named according to the name of the input feature class" \  
 "and the species selected for prediction."  
 self.canRunInBackground = True  
  
 def getParameterInfo(self):  
 *"""Define parameter definitions"""* param0 = arcpy.Parameter(  
 displayName="Input Training Features",  
 name="in\_fc",  
 datatype="DEFeatureClass",  
 parameterType="Required",  
 direction="Input")  
  
 param1 = arcpy.Parameter(  
 displayName="Variables to Predict",  
 name="sp\_field",  
 datatype="Field",  
 parameterType="Required",  
 direction="Input",  
 multiValue=True)  
  
 param1.parameterDependencies = [param0.name]  
  
 param2 = arcpy.Parameter(  
 displayName="Output Workspace",  
 name="out\_ws",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
  
 param3 = arcpy.Parameter(  
 displayName="Output Files Prefix",  
 name="prefix",  
 datatype="GPString",  
 parameterType="Required",  
 direction="Input")  
  
 param4 = arcpy.Parameter(  
 displayName="Number of Trees",  
 name="n\_tree",  
 datatype="GPLong",  
 parameterType="Required",  
 direction="Input")  
  
 param5 = arcpy.Parameter(  
 displayName="# of Random Variables - mtry",  
 name="m\_try",  
 datatype="GPLong",  
 parameterType="Required",  
 direction="Input")  
  
 param6 = arcpy.Parameter(  
 displayName="Compensate for Sparse Categories",  
 name="sparse",  
 datatype="GPBoolean",  
 parameterType="Required",  
 direction="Input")  
 param6.value = True  
  
 param7 = arcpy.Parameter(  
 displayName="Percent Excluded for Validation",  
 name="pct\_train",  
 datatype="GPLong",  
 parameterType="Required",  
 direction="Input")  
 param7.filter.type = "ValueList"  
 param7.filter.list = [10, 20, 30]  
  
 param8 = arcpy.Parameter(  
 displayName="Number of Validation Runs",  
 name="n\_validation",  
 datatype="GPLong",  
 parameterType="Required",  
 direction="Input")  
  
 # Fifth parameter  
 param9 = arcpy.Parameter(  
 displayName="Select Explanatory Rasters",  
 name="in\_rasters",  
 datatype="DERasterDataset",  
 parameterType="Required",  
 direction="Input",  
 multiValue=True)  
  
  
 parameters = [param0, param1, param2, param3, param4, param5,  
 param6, param7, param8, param9]  
  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
  
 return  
  
 def execute(self, parameters, messages):  
 ws\_fishnet = r"C:\Users\hashimotoy\Desktop\ws\_prep\fishnet.gdb"  
  
 in\_fc = parameters[0].valueAsText  
 lst\_sp = parameters[1].valueAsText  
 out\_ws = parameters[2].valueAsText  
 prefix = parameters[3].valueAsText  
 n\_tree = parameters[4].valueAsText  
 m\_try = parameters[5].valueAsText  
 sparse = parameters[6].valueAsText  
 pct\_train = parameters[7].valueAsText  
 n\_validation = parameters[8].valueAsText  
 in\_rasters = parameters[9].valueAsText  
  
 prediction\_type = "PREDICT\_RASTER"  
 nm = getBaseName(in\_fc)  
 in\_features = in\_fc  
 arcpy.env.mask = r"C:\Users\hashimotoy\Desktop\ws\_prep\raster\_attributes.gdb\\" + "eco\_" + nm  
  
 lst\_rasters = []  
 lst\_matching = [] # Explanatory raster matching  
 for raster in in\_rasters.split(';'):  
 lst\_rasters.append(raster)  
 sub\_lst = [raster, raster]  
 lst\_matching.append(sub\_lst)  
  
  
  
 treat\_variable\_as\_categorical = None  
 explanatory\_variables = None  
 distance\_features = None  
 explanatory\_rasters = lst\_rasters  
 features\_to\_predict = None  
 explanatory\_variable\_matching = None  
 explanatory\_distance\_matching = None  
 explanatory\_rasters\_matching = lst\_matching  
 use\_raster\_values = True  
 number\_of\_trees = n\_tree  
 minimum\_leaf\_size = None  
 maximum\_level = None  
 sample\_size = None  
 random\_sample = m\_try  
 percentage\_for\_training = pct\_train  
 output\_classification\_table = None  
 compensate\_sparse\_categories = True  
 number\_validation\_runs = n\_validation  
  
 for sp in lst\_sp.split(';'):  
 arcpy.AddMessage("Running forest for " + sp)  
 variable\_predict = sp  
 file\_prefix = prefix + "\_" + nm + "\_" + sp + "\_"  
 output\_features = os.path.join(out\_ws, file\_prefix + "\_output\_features")  
 output\_raster = os.path.join(out\_ws, file\_prefix + "\_rtr")  
 output\_trained\_features = os.path.join(out\_ws, file\_prefix + "\_trained")  
 output\_importance\_table = os.path.join(out\_ws, file\_prefix + "\_varimp")  
 output\_validation\_table = os.path.join(out\_ws, file\_prefix + "\_validation")  
  
 arcpy.stats.Forest(prediction\_type,  
 in\_features,  
 variable\_predict,  
 treat\_variable\_as\_categorical,  
 explanatory\_variables,  
 distance\_features,  
 explanatory\_rasters,  
 features\_to\_predict,  
 output\_features, output\_raster,  
 explanatory\_variable\_matching,  
 explanatory\_distance\_matching,  
 explanatory\_rasters\_matching,  
 output\_trained\_features,  
 output\_importance\_table,  
 use\_raster\_values,  
 number\_of\_trees,  
 minimum\_leaf\_size,  
 maximum\_level,  
 sample\_size,  
 random\_sample,  
 percentage\_for\_training,  
 output\_classification\_table,  
 output\_validation\_table,  
 compensate\_sparse\_categories,  
 number\_validation\_runs)  
  
 return  
  
  
  
class checkRasterForNullValues(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Check Rasters for Null Values"  
 self.description = "Interrogates selected rasters for NULL values"  
 self.canRunInBackground = True  
  
 def getParameterInfo(self):  
 *"""Define parameter definitions"""* # First parameter  
 param0 = arcpy.Parameter(  
 displayName="Select Rasters",  
 name="in\_rasters",  
 datatype="DERasterDataset",  
 parameterType="Required",  
 direction="Input",  
 multiValue=True)  
  
 # Second parameter  
 param1 = arcpy.Parameter(  
 displayName="Output Workspace",  
 name="out\_ws",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
  
 parameters = [param0, param1]  
  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
 lst\_rasters = parameters[0].valueAsText  
 out\_ws = parameters[1].valueAsText  
  
 for raster in lst\_rasters.split(';'):  
 nm = getBaseName(raster)  
 arcpy.AddMessage("Reading {0}".format(raster))  
 outras = IsNull(raster)  
 outras.save("{0}/{1}".format(out\_ws, nm))  
 arcpy.AddMessage("Saving {0}".format(raster))  
 return  
  
  
class setRasterNullValuesTo0Workspace(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Set Null Raster Values to 0 - by Workspace"  
 self.description = "Interrogates rasters in a workspace and checks for null values. If present, NA values are set" \  
 "to 0"  
 self.canRunInBackground = True  
  
 def getParameterInfo(self):  
 *"""Define parameter definitions"""* # First parameter  
 param0 = arcpy.Parameter(  
 displayName="Raster Workspace",  
 name="in\_rasters",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
  
 # Second parameter  
 param1 = arcpy.Parameter(  
 displayName="Output Workspace",  
 name="out\_ws",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
  
 parameters = [param0, param1]  
  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
 in\_ws = parameters[0].valueAsText  
 out\_ws = parameters[1].valueAsText  
 arcpy.env.workspace = in\_ws  
 rasters = arcpy.ListRasters()  
  
 for raster in rasters:  
 nm = getBaseName(raster)  
 arcpy.AddMessage("Reading {0}".format(raster))  
 outras = Con(IsNull(raster), 0, raster)  
 arcpy.AddMessage("Setting null for {0}".format(raster))  
 outras.save("{0}/{1}".format(out\_ws, nm))  
 arcpy.AddMessage("Saving {0}".format(raster))  
 return  
  
class setRasterNullValuesTo0(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Set Raster Null Values to 0 - by Selection"  
 self.description = "Sets all null raster values to 0 in new raster"  
 self.canRunInBackground = True  
  
 def getParameterInfo(self):  
 *"""Define parameter definitions"""* # First parameter  
 param0 = arcpy.Parameter(  
 displayName="Input Rasters",  
 name="in\_rasters",  
 datatype="DERasterDataset",  
 parameterType="Required",  
 direction="Input",  
 multiValue=True)  
  
 param1 = arcpy.Parameter(  
 displayName="Target workspace",  
 name="out\_ws",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
  
 parameters = [param0, param1]  
  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
 lst\_rasters = parameters[0].valueAsText  
 out\_ws = parameters[1].valueAsText  
  
 for raster in lst\_rasters.split(';'):  
 nm = getBaseName(raster)  
 arcpy.AddMessage("Reading {0}".format(raster))  
 outras = Con(IsNull(raster), 0, raster)  
 arcpy.AddMessage("Setting null for {0}".format(raster))  
 outras.save("{0}/{1}".format(out\_ws, nm))  
 arcpy.AddMessage("Saving {0}".format(raster))  
  
 return  
  
class featureToRasterAttributes(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Feature To Raster Attributes"  
 self.description = ""  
 self.canRunInBackground = True  
  
 def getParameterInfo(self):  
 # Define parameter definitions  
  
 # First parameter  
 param0 = arcpy.Parameter(  
 displayName="Input Features",  
 name="in\_features",  
 datatype="DEFeatureClass",  
 parameterType="Required",  
 direction="Input")  
  
 # Second parameter  
 param1 = arcpy.Parameter(  
 displayName="Attribute Field",  
 name="field",  
 datatype="Field",  
 parameterType="Required",  
 direction="Input")  
  
 param1.filter.list = ['TEXT', 'LONG']  
 param1.parameterDependencies = [param0.name]  
  
 param2 = arcpy.Parameter(  
 displayName="Output workspace",  
 name="out\_ws",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input"  
 )  
 param2.defaultEnvironmentName = "workspace"  
  
 parameters = [param0, param1, param2]  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
  
 return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
 import os  
  
 in\_fc = parameters[0].valueAsText  
 in\_field = parameters[1].valueAsText  
 out\_ws = r"C:\Users\hashimotoy\Desktop\ws\_prep\raster.gdb"  
 attributes\_ws = r"C:\Users\hashimotoy\Desktop\ws\_prep\raster\_attributes.gdb"  
 out\_raster = os.path.join(out\_ws, os.path.basename(in\_fc))  
 # Converting to Raster  
 arcpy.AddMessage("Converting to raster: " + out\_raster + "...")  
 raster = arcpy.FeatureToRaster\_conversion(in\_fc, in\_field, os.path.join(out\_ws, out\_raster))  
 # raster = r"C:\Users\hashimotoy\Desktop\ws\_prep\raster.gdb\vri"  
 values = unique\_values(raster, in\_field)  
 # Extract By Attributes each unique value  
 for value in values:  
 if value == "":  
 pass  
 else:  
 print(" Extracting " + value)  
 where\_clause = in\_field + " = '" + value + "'"  
 arcpy.AddMessage(" " + where\_clause)  
 extract = ExtractByAttributes(raster, where\_clause)  
 out\_nm = (in\_field + "\_" + value).lower()  
 extract.save(os.path.join(attributes\_ws, out\_nm))  
 return  
  
class renameDataInWorkspace(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Rename Data Sources in Workspace"  
 self.description = "The tool will batch rename data within a workspace by one of three options: " \  
 "replace, prefix or add suffix. if "  
 self.canRunInBackground = True  
  
 def getParameterInfo(self):  
 *"""Define parameter definitions"""* # First parameter  
 param0 = arcpy.Parameter(  
 displayName="Input Workspace",  
 name="in\_ws",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
 #param0.filter.type = "Workspace"  
  
 param1 = arcpy.Parameter(  
 displayName="Select String Parsing Function",  
 name="parse\_function",  
 datatype="GPString",  
 parameterType="Required",  
 direction="Input")  
  
 param1.filter.list = ["REPLACE SUBSTRING", "ADD PREFIX", "ADD SUFFIX"]  
  
 # Second parameter  
 param2 = arcpy.Parameter(  
 displayName="Input string",  
 name="in\_string",  
 datatype="GPString",  
 parameterType="Required",  
 direction="Input")  
  
 # Third parameter  
 param3 = arcpy.Parameter(  
 displayName="Output string",  
 name="out\_string",  
 datatype="GPString",  
 parameterType="Optional",  
 direction="Input")  
  
 parameters = [param0, param1, param2, param3]  
  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 parameters[3].enabled = False  
 if parameters[1].value:  
 if parameters[1].valueAsText == "REPLACE SUBSTRING":  
 parameters[3].enabled = True  
 else:  
 parameters[3].enabled = False  
 return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
 in\_ws = parameters[0].valueAsText  
 parse\_function = parameters[1].valueAsText  
 in\_string = parameters[2].valueAsText  
 out\_string = parameters[3].valueAsText  
  
 env.workspace = in\_ws  
 fcs = arcpy.ListFeatureClasses()  
 rasters = arcpy.ListRasters()  
  
 lst = fcs + rasters  
  
 for data in lst:  
 nm = getBaseName(data)  
 if in\_string in nm:  
 pass  
 elif parse\_function == "ADD PREFIX":  
 out\_string = in\_string + nm  
 elif parse\_function == "ADD SUFFIX":  
 out\_string = nm + out\_string  
 elif parse\_function == "REPLACE SUBSTRING":  
 if in\_string not in nm:  
 arcpy.AddMessage("\nThe substring " + in\_string) + " is not within data source names"  
 pass  
 else:  
 arcpy.Rename\_management(data, nm.replace(in\_string, out\_string))  
 arcpy.AddMessage("\nRenamed " + nm + " to " + nm.replace(in\_string, out\_string))  
 arcpy.Rename\_management(data, out\_string)  
 arcpy.AddMessage("\nRenamed " + nm + " to " + out\_string)  
 return  
  
class kernelDensityCalculations(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Calculate Kernel Density"  
 self.description = "Tool to generalize vector point and line inputs (use the Focal Statistics tool for raster inputs)"  
 self.canRunInBackground = True  
  
 def getParameterInfo(self):  
 *"""Define parameter definitions"""* """Define parameter definitions"""  
 # First parameter  
 param0 = arcpy.Parameter(  
 displayName="Input Feature Classes",  
 name="lst\_in\_features",  
 datatype="DEFeatureClass",  
 parameterType="Required",  
 direction="Input",  
 multiValue=True)  
  
 param0.filter.list = ["Point", "Multipoint", "Polyline"]  
  
 param1 = arcpy.Parameter(  
 displayName="Search radius",  
 name="search\_radius",  
 datatype="GPLong",  
 parameterType="Required",  
 direction="Input")  
 param1.value = 564 # pi 'r' squared ~ 1sqkm  
  
 param2 = arcpy.Parameter(  
 displayName="Cell size",  
 name="cell\_size",  
 datatype="GPLong",  
 parameterType="Required",  
 direction="Input")  
  
 param2.value = 20 # Default  
  
 param3 = arcpy.Parameter(  
 displayName="Density unit value",  
 name="area\_unit\_scale\_factor",  
 datatype="GPString",  
 parameterType="Required",  
 direction="Input")  
 param3.filter.list = ["SQUARE\_MAP\_UNITS","SQUARE\_KILOMETERS","HECTARES","SQUARE\_METERS"]  
  
 param3.value = "SQUARE\_METERS"  
  
 param4 = arcpy.Parameter(  
 displayName="Output workspace",  
 name="out\_ws",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
  
 parameters = [param0, param1, param2, param3, param4]  
 # parameters = [param0, param1, param2, param3]  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
  
 lst\_inputs = parameters[0].valueAsText  
 search\_radius = parameters[1].valueAsText  
 cell\_size = parameters[2].valueAsText  
 area\_unit\_scale\_factor = parameters[3].valueAsText  
 out\_ws = parameters[4].valueAsText  
  
 for in\_fc in lst\_inputs.split(';'):  
 nm = getBaseName(in\_fc)  
 arcpy.AddMessage("\n\nProcessing " + nm)  
 kd = KernelDensity(in\_features=in\_fc,  
 population\_field="",  
 cell\_size=cell\_size,  
 search\_radius=search\_radius,  
 area\_unit\_scale\_factor=area\_unit\_scale\_factor,  
 out\_cell\_values="DENSITIES",  
 method="PLANAR")  
 kd.save(os.path.join(out\_ws, nm))  
  
 return  
  
class splitRasterByAttributes(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Split Raster by Attributes"  
 self.description = "Output rasters are named according to the attribute field and value"  
 self.canRunInBackground = False  
  
 def getParameterInfo(self):  
 *"""Define parameter definitions"""* # First parameter  
 param0 = arcpy.Parameter(  
 displayName="Input Raster",  
 name="in\_raster",  
 datatype=["DERasterDataset","GPRasterLayer"],  
 parameterType="Required",  
 direction="Input")  
  
 # Second parameter  
 param1 = arcpy.Parameter(  
 displayName="Attribute Field",  
 name="field",  
 datatype="Field",  
 parameterType="Required",  
 direction="Input")  
  
 param1.filter.list = ['TEXT', 'LONG']  
 param1.parameterDependencies = [param0.name]  
  
 param2 = arcpy.Parameter(  
 displayName="Output workspace",  
 name="out\_ws",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input"  
 )  
 param2.defaultEnvironmentName = "workspace"  
  
 parameters = [param0, param1, param2]  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
 in\_raster = parameters[0].valueAsText  
 in\_field = parameters[1].valueAsText  
 out\_ws = parameters[2].valueAsText  
  
 values = unique\_values(in\_raster, in\_field)  
 # Extract By Attributes each unique value  
 for value in values:  
 if value == "":  
 pass  
 else:  
 print(" Extracting " + value)  
 where\_clause = in\_field + " = '" + value + "'"  
 arcpy.AddMessage(" " + where\_clause)  
 extract = ExtractByAttributes(in\_raster, where\_clause)  
 out\_nm = (in\_field + "\_" + value).lower()  
 extract.save(os.path.join(out\_ws, out\_nm))  
 return  
  
class focalStatsByCircle(object):  
 def \_\_init\_\_(self):  
 self.label = "Focal Statistics by Circle"  
 self.description = "Tool to generalize raster inputs (use the Kernel Density tool for vector points and lines)"  
 self.canRunInBackground = True  
  
 def getParameterInfo(self):  
  
 param0 = arcpy.Parameter(  
 displayName="Input Rasters",  
 name="in\_rasters",  
 datatype="DERasterDataset",  
 parameterType="Required",  
 direction="Input",  
 multiValue=True)  
  
  
 param1 = arcpy.Parameter(  
 displayName="Neighbourhood in map units",  
 name="neighbourhood",  
 datatype="GPLong",  
 parameterType="Required",  
 direction="Input")  
 param1.value = 1200 # Circular neighbourhood of 1.2km to account for center of 400m cell grid  
  
 param2 = arcpy.Parameter(  
 displayName="Statistics type",  
 name="stats\_type",  
 datatype="GPString",  
 parameterType="Required",  
 direction="Input",  
 multiValue=False)  
 param2.filter.list = ["MEAN", "MAJORITY", "MAXIMUM", "MEDIAN",  
 "MINIMUM", "MINORITY", "RANGE", "STD",  
 "SUM", "VARIETY"]  
 param2.value = "SUM"  
  
 param3 = arcpy.Parameter(  
 displayName="Output workspace",  
 name="out\_ws",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
  
 parameters = [param0, param1, param2, param3]  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
  
 rasters = parameters[0].valueAsText  
 neighbourhood = parameters[1].valueAsText  
 stats\_type = parameters[2].valueAsText  
 out\_ws = parameters[3].valueAsText  
  
 for in\_raster in rasters.split(';'):  
 arcpy.AddMessage(in\_raster)  
 nm = getBaseName(in\_raster)  
 arcpy.AddMessage("\nProcessing focal statistics for " + nm)  
 fs = FocalStatistics(in\_raster, NbrCircle(neighbourhood, "MAP"),  
 stats\_type,"DATA")  
 fs.save(os.path.join(out\_ws, nm))  
  
 return  
  
class focalStatsByRectangle(object):  
 def \_\_init\_\_(self):  
 self.label = "Focal Statistics by Rectangle"  
 self.description = "Tool to generalize raster inputs (use the Kernel Density tool for vector points and lines)"  
 self.canRunInBackground = True  
  
 def getParameterInfo(self):  
  
 param0 = arcpy.Parameter(  
 displayName="Input Rasters",  
 name="in\_rasters",  
 datatype="DERasterDataset",  
 parameterType="Required",  
 direction="Input",  
 multiValue=True)  
  
  
 param1 = arcpy.Parameter(  
 displayName="Neighbourhood in map units",  
 name="neighbourhood",  
 datatype="GPLong",  
 parameterType="Required",  
 direction="Input")  
 param1.value = 1000 # Rectangular neighbourhood 1km x 1km  
  
 param2 = arcpy.Parameter(  
 displayName="Statistics type",  
 name="stats\_type",  
 datatype="GPString",  
 parameterType="Required",  
 direction="Input",  
 multiValue=True)  
 param2.filter.list = ["MEAN", "MAJORITY", "MAXIMUM", "MEDIAN",  
 "MINIMUM", "MINORITY", "RANGE", "STD",  
 "SUM", "VARIETY"]  
 param2.value = "SUM"  
  
 param3 = arcpy.Parameter(  
 displayName="Output workspace",  
 name="out\_ws",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
  
 parameters = [param0, param1, param2, param3]  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
  
 rasters = parameters[0].valueAsText  
 neighbourhood = parameters[1].valueAsText  
 stats\_type = parameters[2].valueAsText  
 out\_ws = parameters[3].valueAsText  
  
 for in\_raster in rasters.split(';'):  
 arcpy.AddMessage(in\_raster)  
 nm = getBaseName(in\_raster)  
 arcpy.AddMessage("\nProcessing focal statistics for " + nm)  
 fs = FocalStatistics(in\_raster, NbrRectangle(neighbourhood, neighbourhood, "MAP"),  
 stats\_type,"DATA") #  
 fs.save(os.path.join(out\_ws, nm))  
  
 return  
  
class setRasterValuesTo1(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Set Selected Raster Values to 1"  
 self.description = "Sets all raster values to 1 within a specified workspace. For categorical (factor) variables."  
 self.canRunInBackground = False  
  
 def getParameterInfo(self):  
 *"""Define parameter definitions"""* # First parameter  
 param0 = arcpy.Parameter(  
 displayName="Input Rasters",  
 name="in\_rasters",  
 datatype="DERasterDataset",  
 parameterType="Required",  
 direction="Input",  
 multiValue=False)  
  
 param1 = arcpy.Parameter(  
 displayName="Output workspace",  
 name="out\_ws",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
  
 parameters = [param0, param1]  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
 lst\_rasters = parameters[0].valueAsText  
 out\_ws = parameters[1].valueAsText  
  
 for in\_raster in lst\_rasters.split(';'):  
 nm = getBaseName(in\_raster)  
 arcpy.AddMessage("\n" + nm)  
 out\_con = Con(~IsNull(in\_raster), 1, 0)  
 out\_con.save(os.path.join(out\_ws, nm))  
  
 return  
  
  
class pointToRasterGroupByAttributes(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Point to Raster - Group by Attributes"  
 self.description = "Create raster surfaces from one or more attributes in an input point feature class." \  
 " Output rasters will be prefixed by user defined string" \  
 "and appended with the field value. The optional 'group by' parameter will output individual rasters" \  
 "based on attribute."  
 self.canRunInBackground = True  
  
 def getParameterInfo(self):  
 *"""Define parameter definitions"""* # First parameter  
 param0 = arcpy.Parameter(  
 displayName="Input Point Feature Class",  
 name="in\_pts",  
 datatype="GPFeatureLayer",  
 parameterType="Required",  
 direction="Input")  
  
 # Second parameter  
 param1 = arcpy.Parameter(  
 displayName="Attribute fields",  
 name="fields",  
 datatype="Field",  
 parameterType="Required",  
 direction="Input",  
 multiValue=True)  
 param1.parameterDependencies = [param0.name]  
  
 # Third parameter  
 param2 = arcpy.Parameter(  
 displayName="Output Workspace",  
 name="out\_ws",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
 param2.defaultEnvironmentName = "workspace"  
 # param2.filter.list = ["Local Database"]  
  
 # Fourth parameter  
 param3 = arcpy.Parameter(  
 displayName="Prefix for out raster name",  
 name="prefix",  
 datatype="GPString",  
 parameterType="Required",  
 direction="Input")  
  
 # Fifth parameter  
 param4 = arcpy.Parameter(  
 displayName="Cell Assignment",  
 name="assignment\_type",  
 datatype="GPString",  
 parameterType="Required",  
 direction="Input")  
 param4.filter.type = 'ValueList'  
 param4.filter.list = ["MEAN", "MAXIMUM", "MOST\_FREQUENT",  
 "MINIMUM","RANGE", "STANDARD\_DEVIATION",  
 "SUM", "COUNT"]  
 param4.value = "MEAN"  
  
 # Sixth parameter  
 param5 = arcpy.Parameter(  
 displayName="Cell Size",  
 name="cell\_size",  
 datatype="GPLong",  
 parameterType="Input",  
 direction="Input")  
 param5.value = 400  
  
 # Seventh Optional parameter  
 param6 = arcpy.Parameter(  
 displayName="Optional - Group By attribute",  
 name="groupby",  
 datatype="Field",  
 parameterType="Optional",  
 direction="Input",  
 multiValue = False)  
  
 param6.parameterDependencies = [param0.name]  
  
 parameters = [param0, param1, param2, param3, param4, param5, param6]  
  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
 # Set environment settings  
  
 in\_pts = parameters[0].valueAsText  
 fields = parameters[1].valueAsText  
 out\_ws = parameters[2].valueAsText  
 prefix = parameters[3].valueAsText  
 assignment\_type = parameters[4].valueAsText  
 cell\_size = parameters[5].valueAsText  
 groupby = parameters[6].valueAsText  
  
 env.workspace = out\_ws  
  
 arcpy.AddMessage("Running Point to Raster on: " + in\_pts)  
  
 for field in fields.split(';'):  
 valField = field  
  
 if groupby is None:  
 arcpy.AddMessage("\n Processing " + field)  
 inFeatures = in\_pts  
 outRaster = (prefix + "\_" + field).lower()  
 arcpy.PointToRaster\_conversion(inFeatures, valField, outRaster,  
 assignment\_type, "", cell\_size)  
  
 else:  
 values = unique\_values(in\_pts, groupby)  
 for value in values:  
 arcpy.AddMessage("\n Processing " + field + " for " + str(value))  
 nm = getBaseName(in\_pts)  
 out\_nm = nm + "\_" + str(value)  
 where\_clause = groupby + " = '" + str(value) + "'"  
 inFeatures = arcpy.Select\_analysis(in\_pts, out\_nm, where\_clause)  
 outRaster = (prefix + "\_" + field + "\_" + str(value)).lower()  
 arcpy.PointToRaster\_conversion(inFeatures, valField, outRaster,  
 assignment\_type, "", cell\_size)  
  
  
 return  
  
class getColumnIndex(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Get Column Index"  
 self.description = "Using pandas module, read a csv file and get the index of selected columns"  
 self.canRunInBackground = True  
  
 def getParameterInfo(self):  
 *"""Define parameter definitions"""* # First parameter  
 param0 = arcpy.Parameter(  
 displayName="Input CSV",  
 name="csv",  
 datatype="DEFile",  
 parameterType="Required",  
 direction="Input")  
 param0.filter.list = ['txt', 'csv']  
  
 # Second parameter  
 param1 = arcpy.Parameter(  
 displayName="Fields",  
 name="fields",  
 datatype="Field",  
 parameterType="Required",  
 direction="Input",  
 multiValue=True)  
 param1.parameterDependencies = [param0.name]  
  
 parameters = [param0, param1]  
  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
 import pandas as pd  
  
 in\_csv = parameters[0].valueAsText  
 fields = parameters[1].valueAsText  
  
 arcpy.AddMessage("\n\nGetting indices for selected columns...\n\n")  
 csv = pd.read\_csv(in\_csv)  
  
 for field in fields.split(';'):  
 index = csv.columns.get\_loc(field)  
 arcpy.AddMessage("\n " + field + " : " + str(index) + "\n")  
  
 return  
  
class extractClimateValuesForRange(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Extract Multi Values from Climate Data by Temporal Range"  
 self.description = "The tool will extract the values of selected climate variables for" \  
 "the input points for each year in the study range 2007-2017. Basename of filename" \  
 "will be appended with \_annual\_clim. Note intermediate files will be output to the 'Output" \  
 "workspace'"  
 self.canRunInBackground = True  
  
 def getParameterInfo(self):  
 *"""Define parameter definitions"""* # First parameter  
 param0 = arcpy.Parameter(  
 displayName="Input Features",  
 name="in\_features",  
 datatype="GPFeatureLayer",  
 parameterType="Required",  
 direction="Input")  
  
 # Second parameter  
 param1 = arcpy.Parameter(  
 displayName="Output Workspace",  
 name="out\_ws",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
  
 # Third parameter  
 param2 = arcpy.Parameter(  
 displayName="Select climate variables",  
 name="clim\_var",  
 datatype="GPString",  
 parameterType="Input",  
 direction="Input",  
 multiValue=True)  
 param2.filter.type = 'ValueList'  
 param2.filter.list = ["ahm", "bffp","cmd", "dd5", "dd18", "effp", "mat", "map",  
 "ppt", "pas\_sp", "pas\_wt", "shm", "tave\_sp", "tave\_wt", "tave04"]  
  
 # Fourth parameter  
 param3 = arcpy.Parameter(  
 displayName="Workspace containing climate rasters",  
 name="clim\_ws",  
 datatype="DEWorkspace",  
 parameterType="Input",  
 direction="Input")  
  
 parameters = [param0, param1, param2, param3]  
  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
 pts = parameters[0].valueAsText  
 out\_ws = parameters[1].valueAsText  
 clim\_var = parameters[2].valueAsText  
 ws\_clim = parameters[3].valueAsText  
  
 base\_nm = getBaseName(pts)  
  
 lst\_merge = []  
 for i in range(2007,2018):  
 out\_fc = os.path.join(out\_ws,base\_nm + "clim\_" + str(i))  
 arcpy.AddMessage("\nRunning value extraction for " + out\_fc)  
 in\_pts = arcpy.CopyFeatures\_management(pts, out\_fc)  
 checkFieldExists(in\_pts, "year\_txt", "TEXT")  
 lst\_clim = []  
 for var in clim\_var.split(';'):  
 clim\_raster = os.path.join(ws\_clim,"clim\_" + var + "\_" + str(i))  
 arcpy.AddMessage(" " + clim\_raster)  
 lst\_clim.append([clim\_raster, var])  
 arcpy.CalculateField\_management(out\_fc, "year\_txt", "'" + str(i) + "'")  
 ExtractMultiValuesToPoints(out\_fc, lst\_clim)  
 lst\_merge.append(out\_fc)  
 arcpy.AddMessage("\n\nCompleted extractions by year. \n\nMerging into final dataset...")  
 arcpy.Merge\_management(lst\_merge, os.path.join(out\_ws, base\_nm + "\_annual\_clim"))  
 return  
  
class createPredictedDensitySurfaces(object):  
 def \_\_init\_\_(self):  
 *"""Define the tool (tool name is the name of the class)."""* self.label = "Create Species Response Densities"  
 self.description = "This tool will create density surfaces for predicted SDMs output from 'cforest'"  
 self.canRunInBackground = False  
  
 def getParameterInfo(self):  
 *"""Define parameter definitions"""* # First parameter  
 param0 = arcpy.Parameter(  
 displayName="Input Directory of CSV response files",  
 name="in\_csv",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
  
 # Second parameter  
 param1 = arcpy.Parameter(  
 displayName="Geodatabase of fishnets for each ecosection",  
 name="ws\_fishnet",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
  
  
 # Third parameter  
 param2 = arcpy.Parameter(  
 displayName="Output Workspace",  
 name="ws\_out",  
 datatype="DEWorkspace",  
 parameterType="Required",  
 direction="Input")  
  
 # Fourth parameter  
 param3 = arcpy.Parameter(  
 displayName="Snap Raster",  
 name="snap\_raster",  
 datatype="DERasterDataset",  
 parameterType="Required",  
 direction="Input")  
  
 parameters = [param0, param1, param2, param3]  
  
 return parameters  
  
 def isLicensed(self):  
 *"""Set whether tool is licensed to execute."""* return True  
  
 def updateParameters(self, parameters):  
 *"""Modify the values and properties of parameters before internal  
 validation is performed. This method is called whenever a parameter  
 has been changed."""* return  
  
 def updateMessages(self, parameters):  
 *"""Modify the messages created by internal validation for each tool  
 parameter. This method is called after internal validation."""* return  
  
 def execute(self, parameters, messages):  
 ws\_response = parameters[0].valueAsText  
 ws\_fishnet = parameters[1].valueAsText  
 ws\_output = parameters[2].valueAsText  
 snap\_raster = parameters[3].valueAsText  
  
 # Loop through the CSV files of predicted densities  
 # For each CSV file, get the basename, join it to the appropriate ecosection fishnet  
 # Output a raster  
 env.workspace = ws\_response  
 arcpy.env.snapRaster = snap\_raster  
 files = arcpy.ListFiles("\*.csv")  
 for f in files:  
 f\_nm = getBaseName(f)  
 arcpy.AddMessage(f\_nm)  
 eco = f\_nm[7:10]  
 sp = f\_nm[11:len(f\_nm)]  
 arcpy.AddMessage("Preparing " + sp.upper() + " for " + eco + "...")  
 fishnet\_lyr = arcpy.MakeFeatureLayer\_management(os.path.join(ws\_fishnet,eco), "fishnet\_lyr")  
 joined = arcpy.AddJoin\_management(fishnet\_lyr, "id\_fishnet", f, "id\_fishnet")  
 arcpy.FeatureToRaster\_conversion(joined, sp, os.path.join(ws\_output, eco + "\_" + sp), cell\_size=400)  
 arcpy.AddMessage("\n\nCompleted rasterized densities. Check results in " + ws\_output)  
 return

APPENDIX 4 – R Scripts

Geoprocessing Issues:

Factor levels present within the prediction dataset not present within the training e.g. 14 levels of BEC Subzones with only 13 in the Training set.

APPENDIX C – R Functions

# Create a data frame from a list of csv files converted to a list of data frames merged by common column values

path <- "C:/Users/hashimotoy/Documents/MGIS/Analysis/Models/grid\_cell/cab/glmm\_2007to2017"

extension = "\*.csv"

batch\_read <- function(path, extension) {

file\_names <- list.files(path, pattern = extension)

data\_list <- lapply(paste(path, file\_names, sep = "/"), read.csv)

data\_frame <- bind\_rows(data\_list)

data\_frame

}

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

# From a list of CSV files, select columns and merge

# Helper function

readdata <- function(filename) {

df <- read.csv(filename)

vec <- df[, 4]# Identify the columns

names(vec) <- df[, 1]

return(vec)

}

result <- do.call(rbind, lapply(filenames, readdata))

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

# Merge all files in a folder

wd = "C:/Users/hashimotoy/Documents/MGIS/Data/climateBC/climatebc\_normals1980-2010"

setwd(wd)

files <- list.files(pattern = "\\.csv")

all\_files <- lapply(files, function(x){

read.csv(x, header=TRUE)

})

df <- do.call(rbind.data.frame, all\_files)

write.csv(df, "climatebc\_normals\_1980-2010\_merged.csv", row.names = FALSE)

# Another way

library(dplyr)

library(readr)

df <- do.call(rbind(lapply(files, function(x)read.table(x, header=TRUE, sep = ","))))

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

# Flat correlation matrix with P and r values

library(tidyr)

library(tibble)

library(Hmisc)

flat\_cor\_mat <- function(cor\_r, cor\_p){

#This function provides a simple formatting of a correlation matrix

#into a table with 4 columns containing :

# Column 1 : row names (variable 1 for the correlation test)

# Column 2 : column names (variable 2 for the correlation test)

# Column 3 : the correlation coefficients

# Column 4 : the p-values of the correlations

library(tidyr)

library(tibble)

cor\_r <- rownames\_to\_column(as.data.frame(cor\_r), var = "row")

cor\_r <- gather(cor\_r, column, cor, -1)

cor\_p <- rownames\_to\_column(as.data.frame(cor\_p), var = "row")

cor\_p <- gather(cor\_p, column, p, -1)

cor\_p\_matrix <- left\_join(cor\_r, cor\_p, by = c("row", "column"))

cor\_p\_matrix

}

library(corrplot)

cor <- rcorr(as.matrix(mtcars[, 1:7]))

my\_cor\_matrix <- flat\_cor\_mat(cor$r, cor$P)

head(my\_cor\_matrix)

corr <- rcorr(as.matrix(df))

corrplot(corr$r, method='number', number.cex= 12/ncol(df\_sub))#To alter font size

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

# Check for NA values

df %>%

select\_if(function(x) any(is.na(x))) %>%

summarise\_each(funs(sum(is.na(.)))) -> extra\_NA

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

# Set NA values to 0

df[is.na(df)] <- 0

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

# The function complete.cases() returns a logical vector indicating which cases are complete.

# list rows of data that have missing values

df[!complete.cases(df),]

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

# Check which version of R is running

Sys.getenv("R\_ARCH")

# "/i386" or "/x64"

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

# UPDATE PACKAGES

update.packages(checkBuilt=TRUE, ask=FALSE)

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

# Find column names with NA values

colnames(df)[colSums(is.na(df)) > 0]

mypath <- path

multmerge = function(mypath){

filenames=list.files(path=mypath, full.names=TRUE)

datalist = lapply(filenames, function(x){read.csv(file=x,header=T)})

Reduce(function(x,y) {merge(x,y)}, datalist)}

mymergeddata = multmerge(path)

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

# UPDATE R Version

# installing/loading the package:

if(!require(installr)) {

install.packages("installr"); require(installr)} #load / install+load installr

library(installr)

# using the package:

updateR()

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

# # Rename a column in R

colnames(data)[colnames(data)=="old\_name"] <- "new\_name

sp\_start <- grep("dabblers", colnames(df))

sp\_end <- grep("tot\_sp", colnames(df))

print(sp\_start)

print(sp\_end)

sp <- colnames(df)[sp\_start:sp\_end]

var\_start <- sp\_end + 1

var\_end <- ncol(df)

print(var\_start)

print(var\_end)

int\_eco <- grep("eco", colnames(df))

int\_trans <- grep("trans\_id", colnames(df))

int\_bec\_zn <- grep("bec\_zone", colnames(df))

int\_bec\_sz <- grep("bec\_sz", colnames(df))

for (i in 1:length(sp)){

nm <- sp[i]

print(nm)

int = grep(nm, colnames(df))

data <- df[c(int,int\_eco,int\_trans,int\_bec\_zn, int\_bec\_sz,var\_start:var\_end)]

colnames(data)[1]<- "sp"

write.csv(data, paste(outwd, paste("rf\_",scale, "\_",nm,".csv", sep = ""), sep = "/"), row.names = FALSE)

}