

# Archaeological Predictive Modeling Guide

MnModel Phase 4

---

Carla Landrum, Elizabeth Hobbs, Alexander Anton, Andrew Brown, and Luke Burds

June 24, 2019

© 2019. The MnModel process and the predictive models it produces are copyrighted by the Minnesota Department of Transportation. Any fair use under copyright law should include the disclaimer above. Any use that extends beyond fair use permitted under copyright law requires the written permission of the Minnesota Department of Transportation.

MnModel was financed with Transportation Enhancement and State Planning and Research funds from the Federal Highway Administration and a Minnesota Department of Transportation match.

# Contents

- Introduction .....5**
  - Prior to Modeling .....5
    - Data Preparation .....5
    - Missing Data .....5
  - Scope.....6
  - Setup .....6
  - Generalized Workflow .....6
  - Using this Guide .....7
    - Text Format .....7
    - Icons.....8
    - Directory Structure and File Names .....8
  - Recording Model Results .....8
- Chapter 1: Data Preparation and Sampling (ArcGIS) .....8**
  - Sample Files .....9
  - Prediction Points..... 10
- Chapter 2: Exploratory Data Analysis and Predictive Modeling (RStudio).....10**
  - Working in R..... 11
    - Create Your Regional R\_DIR Directory ..... 11
    - Copy R Template Scripts..... 11
    - Editing R Scripts ..... 12
  - Start a New Project in RStudio..... 14
    - Open RStudio..... 14
    - Start a New Project..... 15
    - Open R Modeling Code File ..... 15
  - Running Scripts in RStudio ..... 18
    - Setting Paths..... 18
    - Running Code ..... 18
    - R Functions ..... 19

Packages and Libraries .....	19
Documenting Your Project .....	20
Saving Your Project.....	21
Re-Opening a Saved Project .....	21
Overview of R Modeling Files .....	21
Section 1: SETUP .....	21
Load Packages and Libraries.....	22
Select a CRAN Mirror .....	23
Increase Maximum Number of Lines Printed.....	24
Load the Data .....	24
Section 2: DATA FORMATTING AND CLEANING .....	28
Create Response Variables .....	28
Remove Fields Not Necessary for Statistical Modeling.....	31
Calculating NULL Frequencies and Removing NULL Values .....	32
Factor Counts.....	36
Response Variable Counts.....	45
Generating a Call List for Factors.....	46
Section 3: EDA (Exploratory Data Analysis) .....	47
Statistical Decision Matrix .....	47
Summary Statistics (Site Vs Background) .....	47
Kolmogorov-Smirnov Test (K-S Test) .....	48
Assessing Zero and Near Zero Variance .....	51
Histograms.....	53
Skewness (optional).....	57
Collinearity.....	58
Chi-squared Test of Independence.....	64
Section 4: Predictive Model Training and Testing .....	66
Step 1: Generate Training and Testing Datasets .....	66
Step 2: Train a Random Forest Model.....	69
Step 3. Optimize Model Fitting Parameters .....	70
Step 4: Assess Model Performance .....	76

Step 5: Complete Four Preliminary Models .....	85
Step 6: Fit a Final Model to the Entire Dataset .....	85
Section 5: Generate Region-wide Model Predictions .....	86
Section 6: Exporting Model Predictions in .CSV Format .....	86
Section 7: ‘Knitting’ R Script to .html .....	87
<b>Chapter 3: Rasterization and Model Classification (ArcGIS) .....</b>	<b>88</b>
Section 1: Rasterization .....	88
Create Floating Point Rasters .....	88
Section 2: Model Classification .....	88
‘Maximum Accuracy’ Models .....	89
Target Sensitivity Models .....	90
Section 3: Model Evaluation .....	92
Extract Phase 3 Models for Comparison .....	92
Sample Models .....	92
Evaluate ‘Maximum Accuracy’ Model Performance .....	93
Section 4: Survey Implementation Model .....	94
Make a Maximum Accuracy Survey Implementation Model .....	94
Make Target Sensitivity Survey Implementation Models .....	95
Section 5: Create Statewide Models.....	95
<b>Conclusions .....</b>	<b>96</b>
<b>References.....</b>	<b>96</b>
<b>Appendix A: Software Installation .....</b>	<b>97</b>
<b>Appendix B: Data Preparation and Sampling.....</b>	<b>97</b>
<b>Appendix C: Tools Handbook.....</b>	<b>97</b>

# Introduction

This User's Guide is written for non-statisticians and presents workflow procedures and R Scripts for MnModel Phase 4 archaeological predictive modeling (Hobbs 2019b). Procedures for developing the MnModel Phase 4 Historic Vegetation Model are documented separately (Landrum and Hobbs 2019). [Appendix A](#) of this Guide provides higher-level instructions for download and installation of R and RStudio. These are the steps that are performed once per computer. [Appendix B](#) of the Guide covers preparation of archaeological data for sampling and implementing ArcGIS Python sampling tools. [Appendix C](#) of the Guide catalogs the tools developed for this project and describes their use. The main sections of this document cover executing statistical modeling and predictions in RStudio, exporting the results to ArcGIS, and classifying and evaluating the models. These steps are repeated for each modeling region.

## Prior to Modeling

---

### Data Preparation

Both archaeological data (site and survey locations) and environmental data (for predictor variables) are required. Minnesota is fortunate to have a large quantity of useful, high resolution data available in GIS format. However, most data reflect modern conditions. A considerable amount of work was required to use the available data to model historic/prehistoric environments. This work included conditioning the Digital Terrain Model (DTM) to reduce the effects of modern anthropogenic features such as roads and ditches; modeling historic and prehistoric hydrography; and modeling historic vegetation (Hobbs 2019a, 2019b; Hobbs et al. 2019a).

For any of the models we run, we use a suite of environmental variables as 'predictor' variables. The site and survey models should use the same predictors. A discussion of the modeling variables can be found Hobbs, Walsh and Hudak (2019). The [Appendix C](#) of this Guide documents the customized ArcGIS tools used to create the variable rasters.

Finally, the archaeological data are prepared and used to sample the predictor variables. This step generates the data tables that are used for statistical analysis and modeling. These procedures are detailed in [Appendix B, \*Preparing Data for Modeling\*](#).

### Missing Data

Statistical procedures will not run on data with NULL values. The gSSURGO soils data for Minnesota are not complete. Moreover, the data contain NULL values for most variables where water bodies or disturbed areas are present. Because the soils data are useful for most of the modeled area, we use a mask to denote where soil variables are most likely to have NULL values. This allows us to create two sets of sample points: one with soil variables and one without. We then run two version of each model, one using soil variables, for locations that have no NULL values, and a separate version without soil variables for the entire region. The final model will be a composite of the two models. Please refer to [Appendix B](#) for more information about the modeling mask.

## Scope

---

It is beyond the scope of this technical User's Manual to teach the concepts or theory behind the statistical applications herein. The User's Guide will not make the user proficient in statistics. This scope of work is pursuant to Oehlert and Shea's (Oehlert and Shea 2007a; 2007b) work in addition to the Pennsylvania Department of Transportation's methods for archeological modeling (Harris et al. 2015).

R is also referred to as RStudio throughout this User's Guide. This User's Guide familiarizes the User with coding, however, will likely not make the User proficient in R coding. There are multiple sources of information, including online forums, books and other published materials to teach R coding. The User should explore these resources to get the most out of R.

## Setup

---

Prior to beginning any of the work described in this manual, the User must install the necessary software, configure the computer, and set up data directories. [Appendix A](#) provides an outline for: 1) downloading and installing R and 2) downloading and installing R Studio. The remainder of this User's Guide assumes these software applications were installed successfully. Please refer to [Appendix A](#) at this point if you have not previously run these procedures or if the computer you are using has not yet been set up for these procedures.

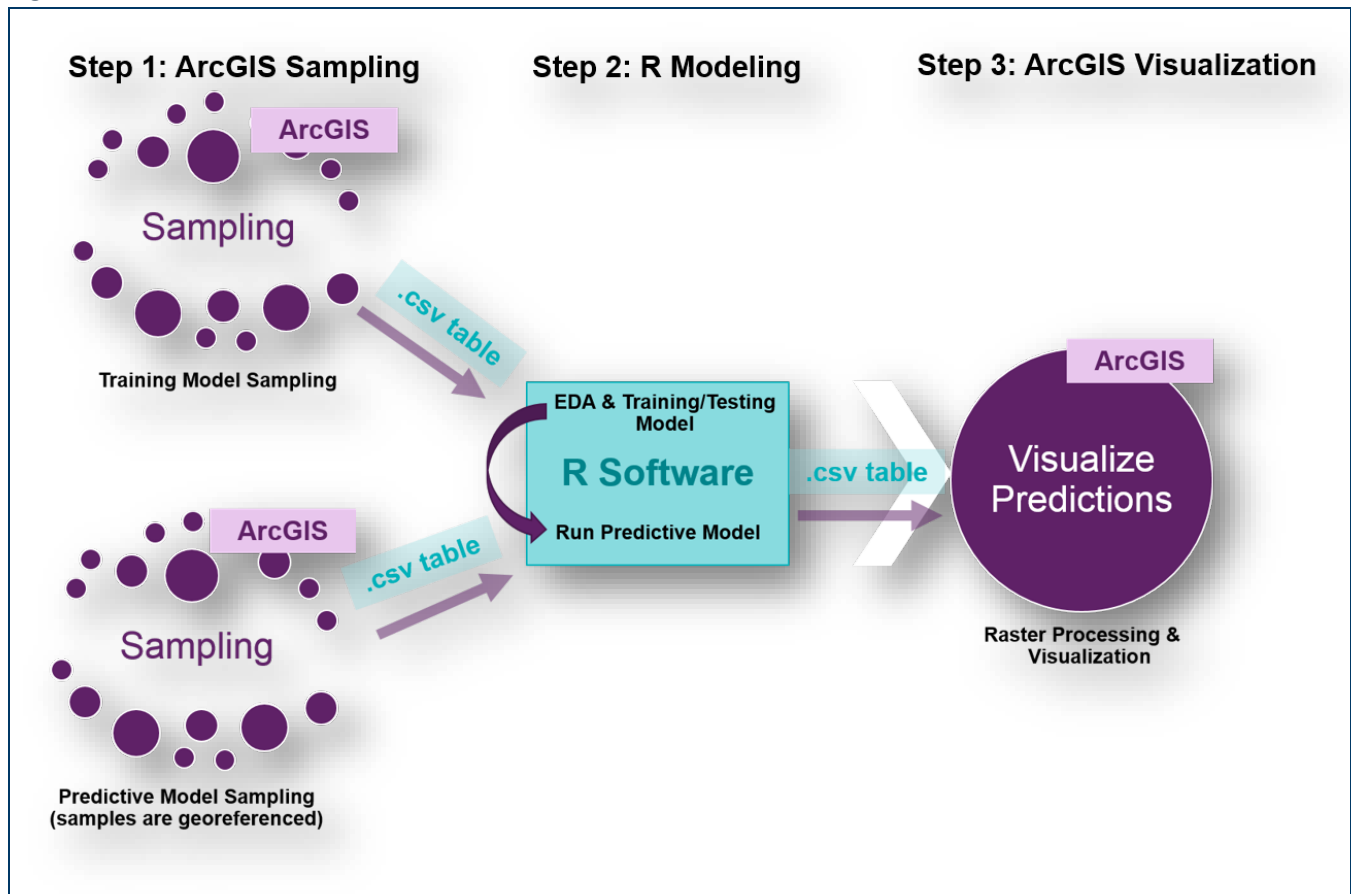
## Generalized Workflow

---

This User Guide details three general steps for completing the predictive modeling: 1) data preparation and sampling, 2) exploratory data analysis (EDA), model training/testing and running model predictions and 3) visualization (rasterization and classification). Figure 1 provides a higher-level overview of these steps. Python tools were developed in ArcGIS to streamline the procedures in Steps 1 and 3.

These three steps are the culmination of a long process of developing GIS data, modeling historic and prehistoric environments, and generating predictor variables. The workflow described in this User Guide does not describe these steps, which must precede sampling.

Figure 1. Generalized Workflow



## Using this Guide





Several conventions have been used in creating this Guide to make it easier to follow.

### Text Format

The following text formats have special meaning:

- **Bold** text generally denotes a tool or command in ArcGIS.
- **Red** text indicates text that needs to be edited for each model run. This may be a region name in a file path, a variable, or some other value either in a file path, file name, or line of code.
- **Blue** text indicates lines of code run in RStudio.
- **Green** text indicates lines of comment in RStudio.
- Variable or field names in ArcGIS tables are enclosed in square brackets []. Variable names in R are not.

## Icons

-  Text with the light bulb icon provide helpful information, hints, and tricks.
-  Text with the 'important' icon indicate things you MUST do.
-  Text with the MS Word icon indicate that you need to record model information in the WORD file you will create for your model record (*TYPE*Report*REG*.docx). You will create this report only if you are using the standard scripts.
-  Text with the MS Excel icon indicate that you should record model information in an Excel spreadsheet record. You will record this information no matter which scripts you are using. Several different spreadsheets were used over the course of the modeling. The user should design a spreadsheet that captures any information that might be needed in the future for displaying and summarizing model results as well as for seeking patterns among the various models run.

## Directory Structure and File Names

A standardized directory structure and file naming conventions are necessary for modeling tools to run efficiently on multiple regions. This User Guide refers to the paths and filenames used in MnModel Phase 4. Anyone adapting the MnModel Phase 4 tools or procedures needs to be aware that using different paths and filenames may require modifying tools accordingly.

## Recording Model Results

---

If using the original R scripts, you will need to copy detailed model results from the R console into a Word document so that they can be preserved. The advantage of using the enhanced R scripts is that R will output all model results in an .html file. This saves quite a bit of time.

Key model results are best recorded in spreadsheets so that they can later be summarized for reporting. Spreadsheets are also useful for keeping checklists. For MnModel Phase 4, the ARCHMOD\_EVAL.xlsx spreadsheet was set up both for recording results of individual models and to allow us to compare model results from one region to another.

## Chapter 1: Data Preparation and Sampling (ArcGIS)

Archaeological predictive modeling is not a simple task. Considerable data preparation is necessary. Both archaeological data (sites and surveys) and environmental data (used as predictor variables) are required. Several documents detail the procedures used to prepare data for MnModel Phase 4. These are available on the [MnModel web site](#).



Once the archaeological database has been prepared and environmental variables created, Chapter 1 procedures generate sample populations for training and running predictive models. These are the final steps necessary to create the data used as input to the statistical analysis. These steps are detailed in [Appendix B](#) of this User Guide and summarized briefly here.

## Sample Files

---

Sample files include both points representing either archaeological sites or surveys and background points where sites/surveys are not known to be found. This allows the statistical procedures to distinguish between the characteristics of places known to have sites/surveys and places where sites/surveys are not known and are assumed to be absent. The sampled value for each site or survey point represents a majority count or mean of raster cell values for each predictor variable falling within the site or survey polygon footprint. [Appendix B](#) details which variables are measured by majority count and which are measured by mean. The sampled value for each background point represents the variable value at that point.

For training a model, sampling consists of attaching the values of each predictor variable (e.g. environmental variables) to georeferenced points representing the response variable (e.g. site, survey, or background point). The response variable contains known information about site presence or survey presence. For a model to recognize a site or survey presence, it is necessary to contrast site or survey locations with background locations, or georeferenced locations where a site or survey has not been confirmed and is assumed absent until proven otherwise. Therefore, the site and survey response variable will include known information about site or survey locations in addition to contrasting background locations. Several ArcGIS sampling tools are available to generate sample datasets (Table 1). [Appendix B](#) discuss the sampling procedures, and the [Appendix C](#) discusses each sampling tool in detail. Instructions in Section 2 of this User Guide detail how to import the sampled data into R.

**Table 1. Sampling Objectives and Corresponding Sampling Tools**

Sampling Objective	Sampling Tool(s)
Sample archeological site polygons to train a predictive model	Batch_Stats <sup>1</sup>
Sample survey polygons to train a predictive model	Batch_Stats <sup>1</sup>
Sample vegetation locations to train a predictive model	Extract Multi Values to Point <sup>2</sup>
Sample background sites to train a predictive model	Create Random Point <sup>2</sup> and Fishnet Grid <sup>2</sup> Extract Multi Values to Point <sup>2</sup>
Sample a prediction grid (30 meter mesh grid) to generate predictions using a trained site, survey or vegetation model	Generate Region Prediction <sup>1</sup>
<sup>1</sup> Custom python tool <sup>2</sup> Standard ArcGIS tool	

## Prediction Points

---

For visualizing statistical models created in R, model predictions are applied to 30 meter mesh of ‘prediction points.’ Prior to modeling in R, these prediction point files must be created containing x,y coordinates for each point as well as sampled values for all predictor variables. Instructions for creating prediction points can be found in [Appendix B](#).

## Chapter 2: Exploratory Data Analysis and Predictive Modeling (RStudio)

Chapter 2 covers loading data into RStudio, data formatting, and statistical evaluations. Exploratory Data Analysis (EDA) is a data diagnostic step to determine if your data are the right type, quantity and quality to generate reliable predictions and to select a defensible model. Arguably, EDA is the most important and time-consuming step in building a defensible predictive model. Finally, you will model your data, then tune and test your models.

## Working in R

---

### Create Your Regional R\_DIR Directory

For each model you run, you will begin a new RStudio Project. The naming conventions and file locations for your projects will be:

- `\MNMODEL4\REGIONS\REG\R_DIR`: This will be the upper level directory for all R files for each region. This directory will have three subdirectories, which will be referred to below as *TYPEMOD*:
  - `SITEMOD`
  - `SURVMOD`
  - `VEGMOD`

Within each of these *TYPEMOD* subdirectories, you will create additional subdirectories for each model version you construct (`ALLSITE1_V1`, `ALLSITE1_V2`, `SOILSITE1_V1`, etc.). You should expect to have more than one version of each model, as the first modeling run is likely to show you where you can make some improvements. The names of these directories will be referred to below as *TYPE\_VN*. At a minimum, you should have the following folders:

- `\MNMODEL4\REGIONS\REG\R_DIR\SITEMOD\ALLSITE1_V1`
- `\MNMODEL4\REGIONS\REG\R_DIR\SITEMOD\ALLSURV1_V1`
- `\MNMODEL4\REGIONS\REG\R_DIR\SITEMOD\SOILSITE1_V1`
- `\MNMODEL4\REGIONS\REG\R_DIR\SITEMOD\SOILSURV1_V1`



Each of these folders will be an R ‘working directory’. Your R scripts and all output files for a single model will be saved in each working directory.

### Copy R Template Scripts

R modeling code template files have been created for the basic predictive model options that we will run. There are two types of scripts available. They run the same modeling procedures but differ in how they are used and in how the results are documented.

Standard R scripts, the original scripts developed for MnModel Phase 4, run in RStudio without the benefit of R Markdown. Using these scripts requires separate documentation of model results. The enhanced scripts require R v. 3.5 (or newer) and RStudio v1.2 (or newer) and use R Markdown to document both modeling procedures and results.

All scripts are in the `\MNMModel4\TOOLS\R\Modeling Scripts\PredictiveModel\TEMPLATE\` directory. The necessary template files are:

- **Standard scripts without jackknife procedures:** Standard scripts without jackknife procedures are in the `\MNMModel4\TOOLS\R\Modeling Scripts\PredictiveModel\TEMPLATE\INITIAL` folder. These are `ALLSITEModelTemplateV2.R`, `ALLSURVModelTemplateV1.R`, `SOILSITEModelTemplateV1.R`, and `SOILSURVModelTemplateV1.R`. Each of these is specific to one of the four models. Be careful to copy each to the correct working directory. These scripts were used for the initial site models and the final survey models in MnModel Phase 4.
- **Standard script with jackknife procedures:** One standard script with jackknife procedures is in the `\MNMModel4\TOOLS\R\Modeling Scripts\PredictiveModel\TEMPLATE\JACKKNIFE` folder. This script (`ALLSITEModelTemplateJack.R`) was used as a prototype to guide development of the enhanced scripts for running the jackknife procedures. It has not been tested. A standard version for the SOILSITE model was not developed.
- **Enhanced scripts without jackknife:** To run models without the jackknife procedure, use the `ModelTemplate.Rmd` and `variables.R` scripts in the `\MNMModel4\TOOLS\R\Modeling Scripts\PredictiveModel\TEMPLATE\RMD\INITIAL` directory. Both need to be copied into each of your working directories. These scripts were used to run intermediate site models and final survey models.
- **Enhanced scripts with jackknife:** To run models using the jackknife procedure, use the `ModelTemplate.Rmd` and `variables.R` scripts in the `\MNMModel4\TOOLS\R\Modeling Scripts\PredictiveModel\TEMPLATE\RMD\JACKKNIFE` directory. Both need to be copied into each of your working directories. These scripts were used to run the final site models. They may also be used to run survey models, but that was not necessary for MnModel Phase 4.

Be sure to save the required templates to your region `R_DIR\MODTYPE\TYPE_VN` working directory prior to running each model. Although the filename doesn't need to change for the `.Rmd` script to run, renaming the `.Rmd` file `TYPE.Rmd` would ensure that it or the associated `.html` file the script produces aren't mistaken for unused templates (for example, `ALLSITE.Rmd`).



**Which version to run?** If you have never used R before – or if you are unfamiliar with the predictive modeling procedures – it would be useful to run a standard script first. This will allow you to examine the code and see the results of each section of code. Running the enhanced scripts is more automated and faster, so once you are familiar with how the procedures work you will enjoy using this version.

## Editing R Scripts

All editing will be done in R.

- **Standard Scripts:** You will need to edit each of the standard scripts as you work through the procedures. Editing will include specifying paths to the input data, the output data, and the working directory. It will also include changing certain modeling parameters. Instructions for these edits are included in this User

Manual. These scripts alert you to text that requires editing by surrounding the text with asterisks. For example, in the code line below you would remove the asterisks and specify the drive letter and path to your sample file.

```
df<-read.csv(file="*driveletter:/path*/MnMODEL4/REGIONS/*REGION*/SAMPLE/ALLSITE.csv",  
header=TRUE,sep=",")
```

- **Enhanced Scripts:** You should not need to edit ModelTemplate.Rmd unless you are changing basic procedures or updating the script to use it with a new version of R. You will edit variables.R for each model in each region. Instructions are provided in this User Manual. In addition, variables.R provides examples of correct lines of code for the data you need to edit.

When editing in R, be aware that:

- Paths to data use slashes differently in R than in Windows or ArcGIS. Sometimes the forward slash (/) indicates a subdirectory change, but in other functions the double back slash (\\) is used. Be sure to follow the conventions in the R script.
- R is case sensitive and will report an error if naming schemes are not exact.
- Be mindful of where quotation marks are required.
- In the User Guide, text in **light blue** indicates functions to run in R. Text in **red** indicates parameters or variables within R functions that must be edited for each model.



**Updating R Syntax:** R will prompt you to update packages regularly and this might require you to update the R script syntax over time. This is normal and should be expected when using R software. For example, the syntax '**as.factor.result=TRUE**' will not run with the most recent 'car' library. The updated syntax for the recent 'car' library is very similar, however, and uses '**as.factor =TRUE**'. You can research the syntax for each function and library under 'Packages' in the RStudio graphical window (or do an online Google search).

# Start a New Project in RStudio

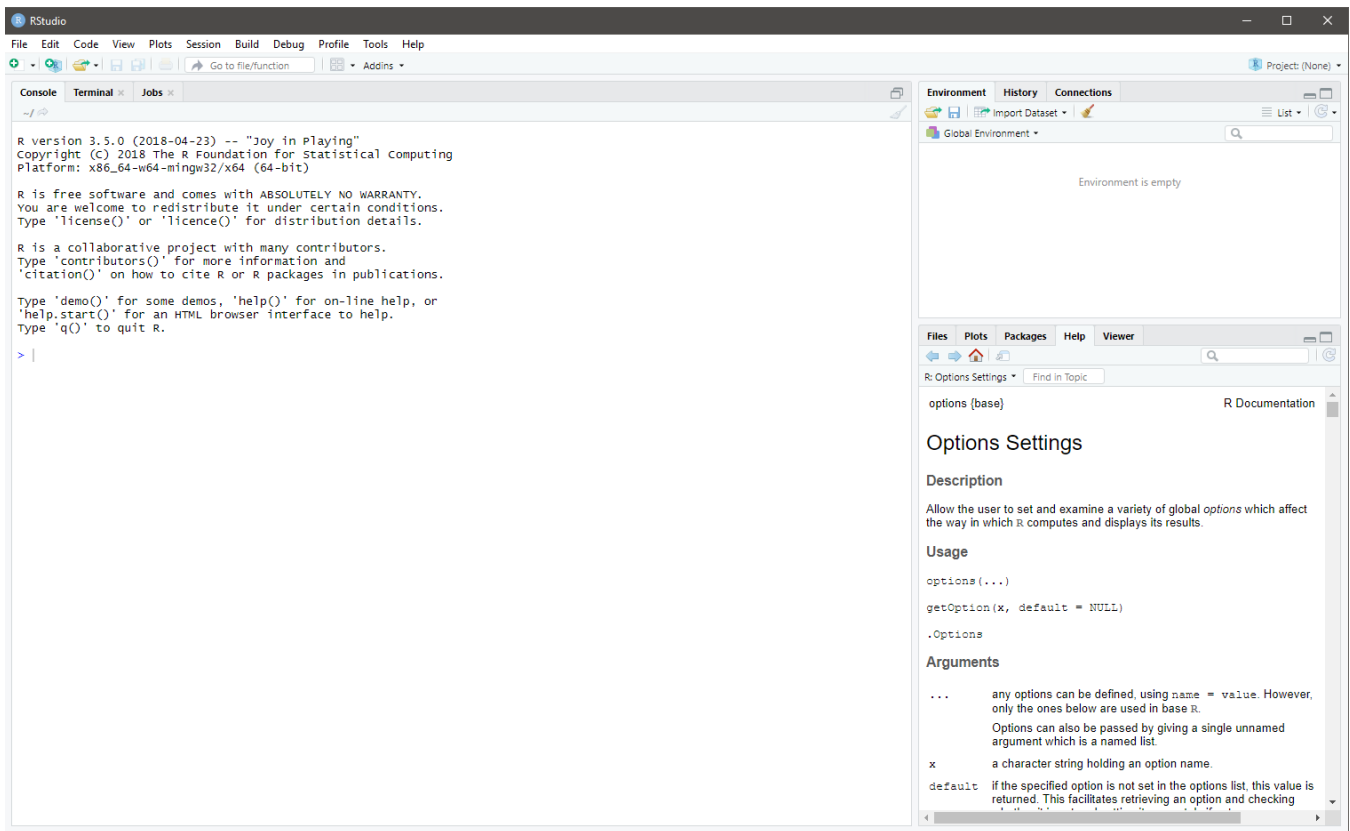
## Open RStudio

Open RStudio from the desktop icon or Start menu of your computer.



You will see a very similar interface to Figure 2. If you do not see this interface, you have opened R and not RStudio.

**Figure 2: RStudio Interface**



**Enhanced Scripts:** If running models using the enhanced script ensure that RStudio-1.2.x is being used in combination with R-3.5.x to allow the enhanced script to have full functionality. Additionally, the following code should be run in the console in RStudio prior to modeling to avoid problems knitting the R script to an .html file.

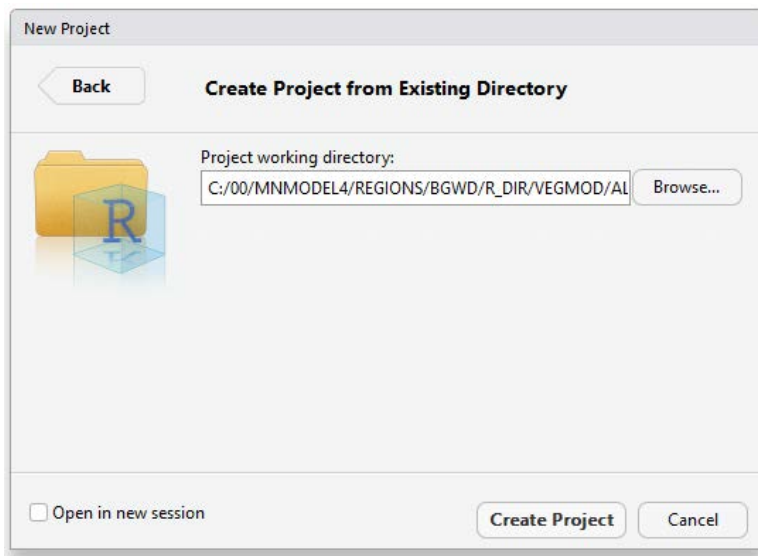
```
install.packages(devtools)
library(devtools)
devtools::install_github("rstudio/reticulate")
```

## Start a New Project

To start a new project:

1. Go to File-> New Project->New Directory ->New Project.
2. Select 'Existing Directory'. It should open the dialog illustrated in Figure 3.
3. Use the Browse button to define the location of  
\\MNMModel4\REGIONS\REG\R\_DIR\TYPEMOD\TYPE\_VN.
4. Click 'Create Project'.

**Figure 3: Create Project from Existing Directory Dialog Box**



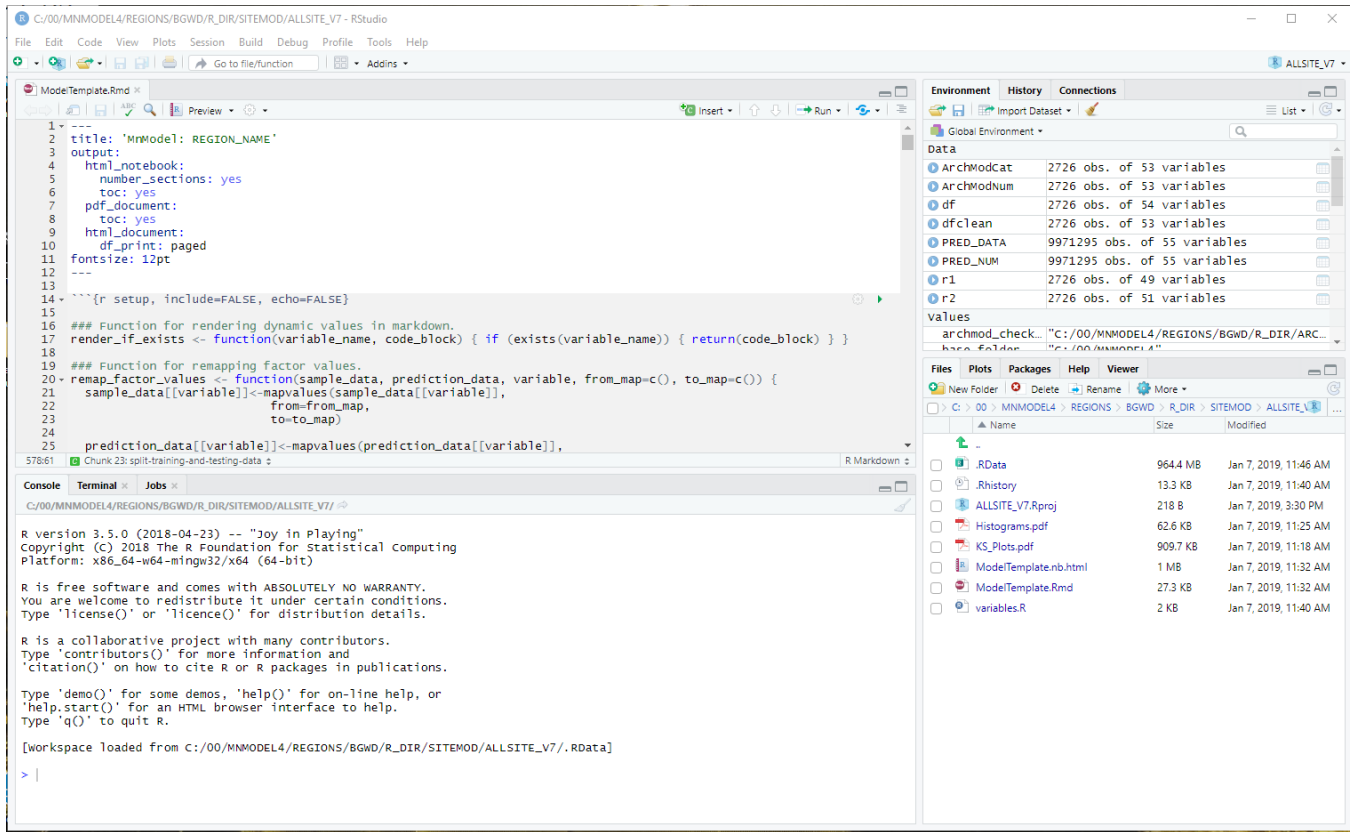
## Open R Modeling Code File

The modeling code templates you need should already have been copied to the working directory you just specified.

- **Standard scripts:** Open the version of *MODTYPE*ModelTemplateVN.R you saved to your working directory.
- **Enhanced script:** Open ModelTemplate.Rmd or *TYPE*.Rmd from your working directory.

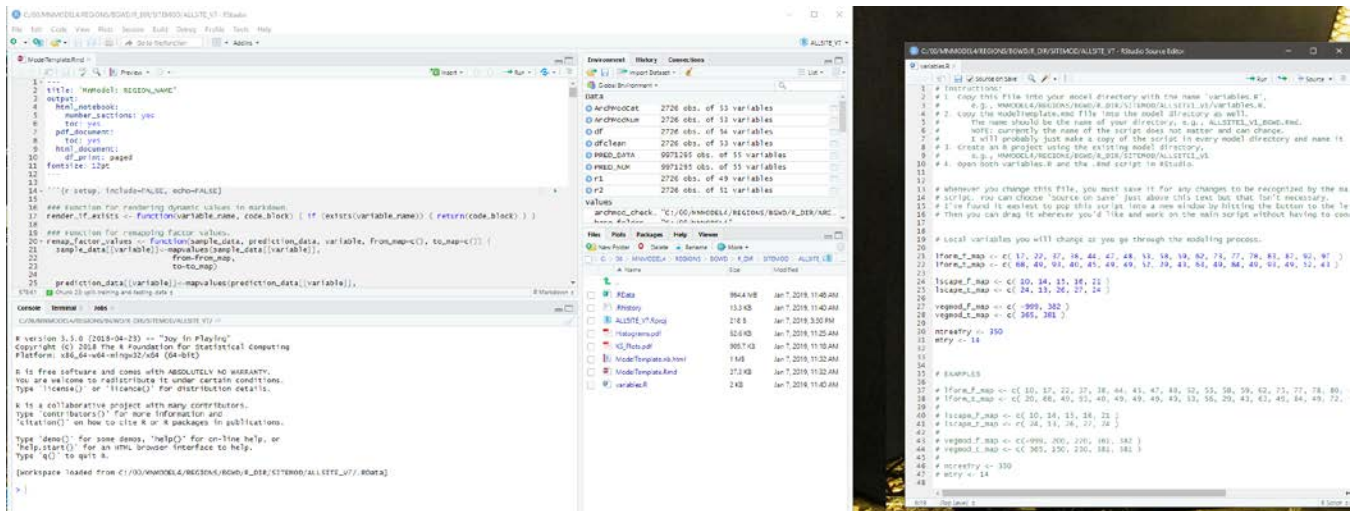
The tool for opening an existing file is the third icon on the RStudio toolbar (Figure 4), or you can select Open File from the File menu. Either should open your working directory and allow you to select your region's modeling code template. When it loads, you should see the code with line numbers along the left side. You will also see a new window, the Console Window, open below the window containing the script (Figure 4).

**Figure 4: Console Window**



The variables.R file can be displayed in a separate window outside of the R application by clicking, holding, and dragging the variables.R tab outside of the R Studio window, creating a separate window for more convenient modeling (Figure 5).

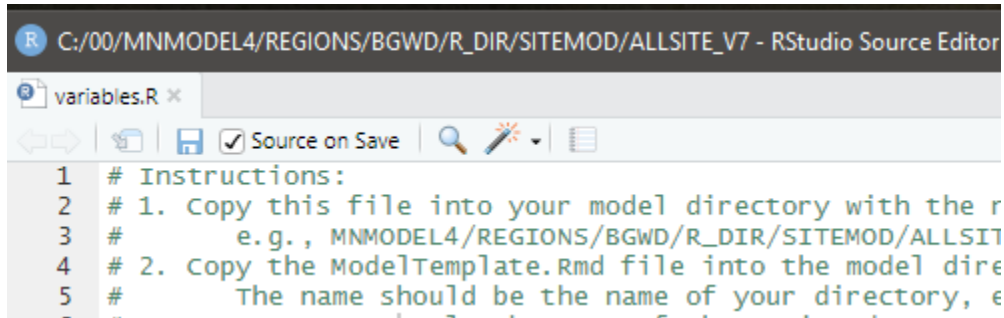
**Figure 5: Separate Editing Window**





Check the 'Source on Save' box at the top of your script window (Figure 6). This will ensure that any changes you made will be saved when you quit your RStudio session or close your project.

**Figure 6: Source on Save**



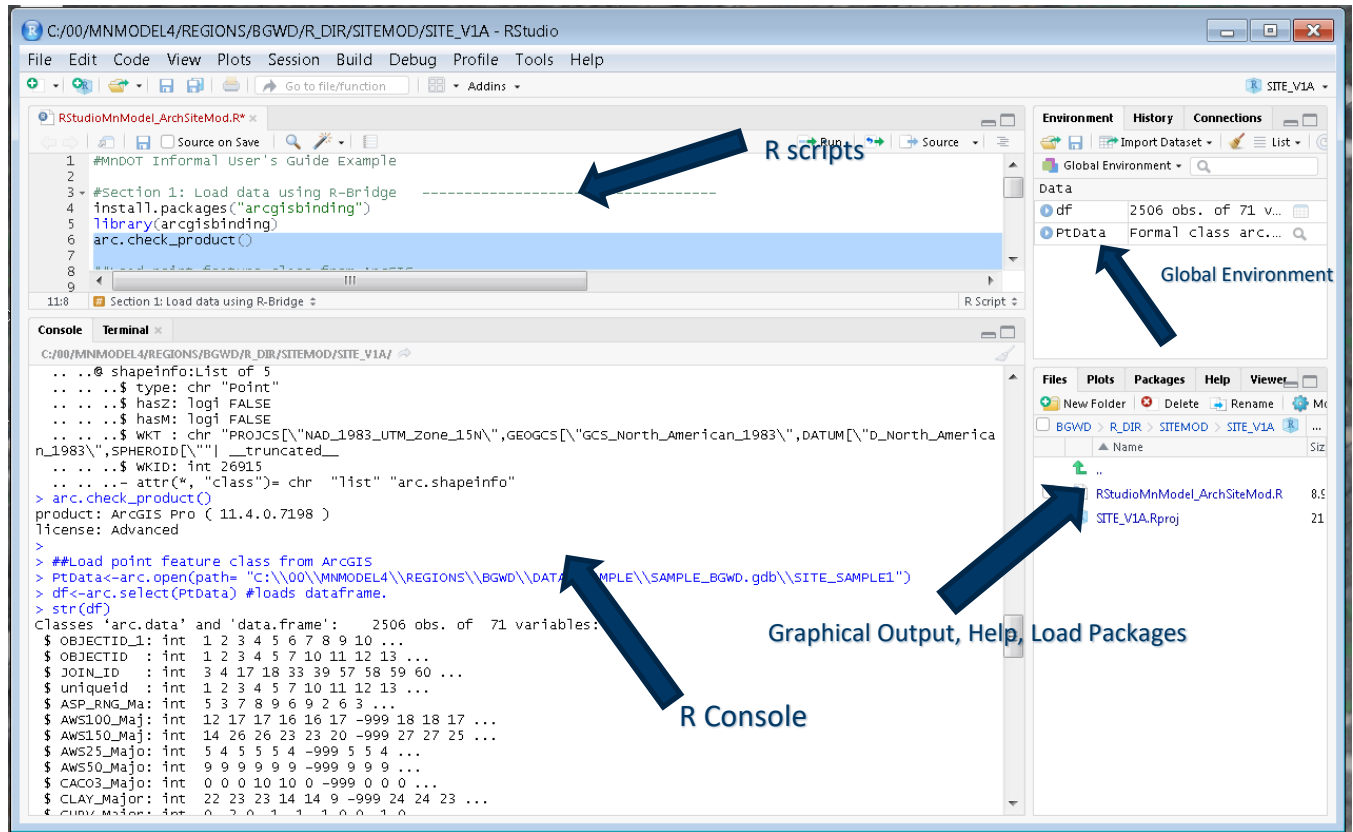
Any modifications to the R script will automatically be saved, so be careful.

Figure 7 introduces the RStudio components.

- R script window:
  - **Standard scripts:** You will run and edit code in the R scripts window.
  - **Enhanced scripts:** You will run chunks of code in the ModelTemplate.Rmd or *TYPE*.Rmd R script window. You will edit variables.R in its own separate script window.
- Global Environment window: The Global Environment window includes your workspace environment and history.
- The Console. The Console will show outputs from running your scripts.
- Graphical Output/Help/Load Packages: This fourth window in the lower right-hand corner of your RStudio window is multi-purpose. It will show you a list of files you have loaded or created, graphs you create, and lists of loaded packages. You may also view help files in this window.

Your RStudio workspace should look very similar to Figure 7.

**Figure 7: RStudio Components**



## Running Scripts in RStudio

### Setting Paths

R will need to know the paths to your data and working directory.

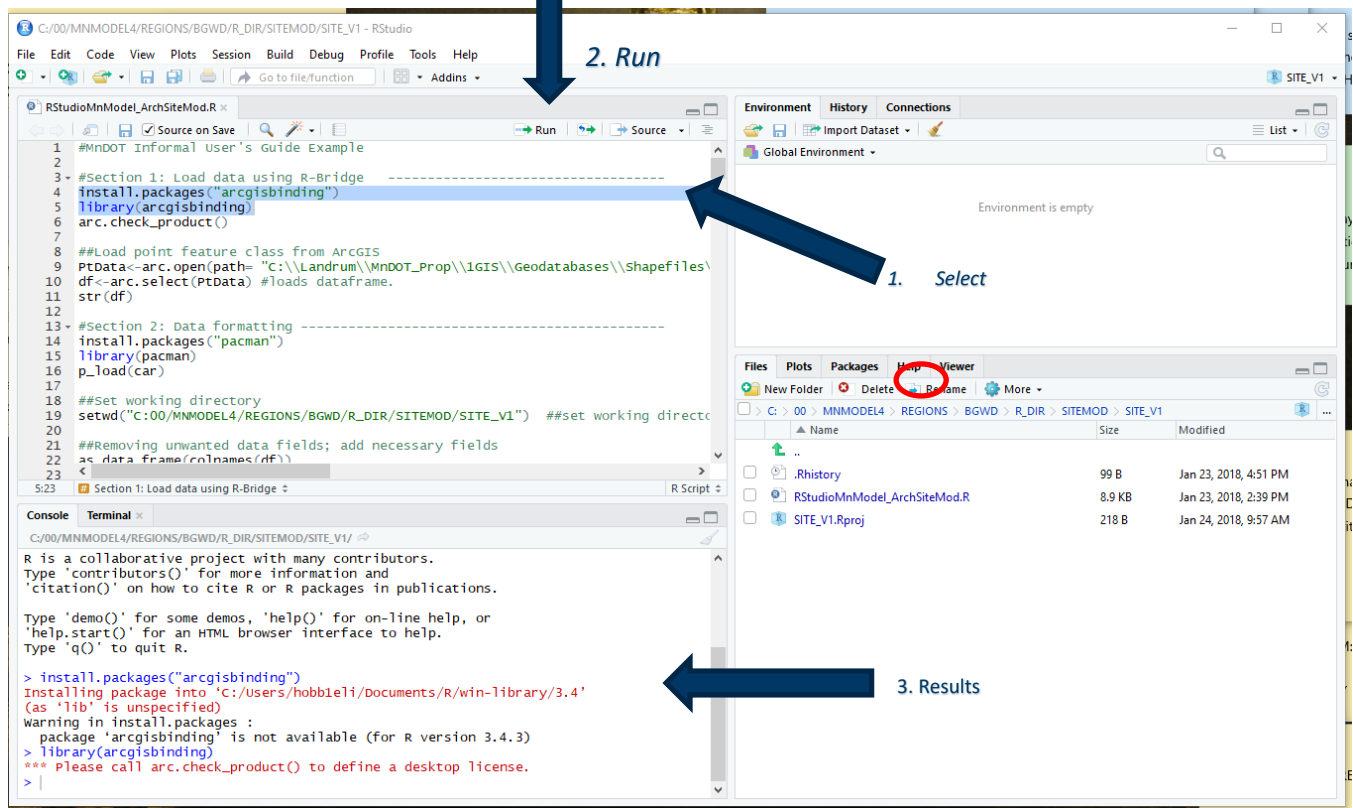
- **Standard scripts:** You will need to edit the first few lines of code to include the correct paths before running the code.
- **Enhanced scripts:** The ModelTemplate.Rmd or *TYPE*.Rmd script should automatically generate paths that correspond to the region and model version you are working on. However, this requires that all directory structure and file naming conventions specified for MnModel Phase 4 have been followed.

### Running Code

Procedures for running scripts differ slightly depending on which version you are using. No matter which type of script you run, you will see the lines of executed script and its results displayed in the Console window.

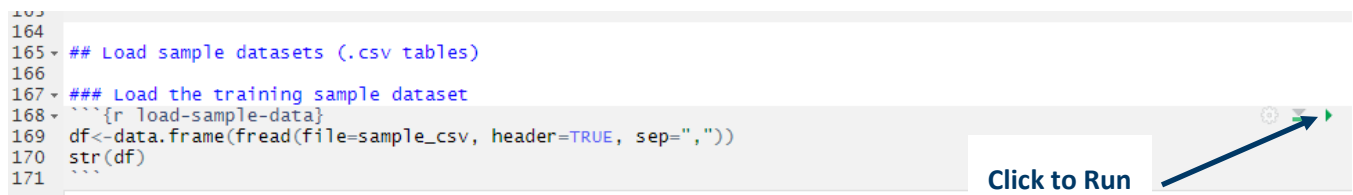
**Standard scripts:** Run code by selecting the lines you want to run in the R script then click ‘Run’ on the toolbar of the script window (Figure 8).

**Figure 8. Running Standard R Script**



**Enhanced scripts:** In the .Rmd file, you will run chunks of code by clicking on the green arrow at the upper right-hand corner of each ‘chunk’ (Figure 9). The gray shaded area in this figure contains the chunk of code to be run. Clicking on the green arrow in the upper right-hand corner of that chunk will run only the lines within the chunk. Arrows on the left, next to the line numbers, will collapse chunks of code.

**Figure 9: Running Code Chunks in .Rmd Scripts**



## R Functions

To learn more about a function, reference the Help tab in the window beneath the Global Environment window (Figure 8). Search for the function’s name and the information should display.

## Packages and Libraries

Package installation is necessary once in your R project, but libraries must be loaded in each session that you need to use them.

- **Standard scripts:** All packages and libraries are loaded in the first part of Section 1 of the scripts.
- **Enhanced scripts:** All packages and libraries are loaded in the first chunk of code in the ModelTemplate.Rmd or *TYPE.Rmd* file.

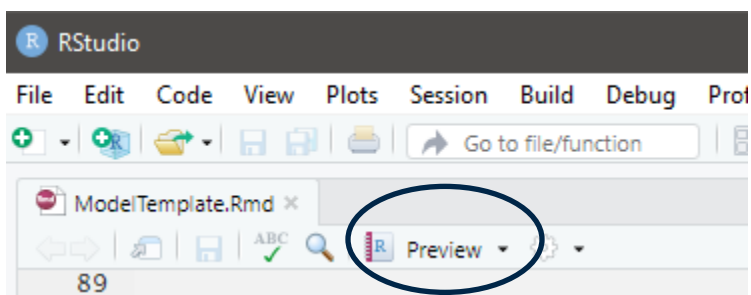
## Documenting Your Project

Perhaps the most noticeable difference between using the standard and enhanced scripts is that the enhanced scripts automatically document your procedures and results while the standard scripts require that you manually copy your results into a WORD document.

**Standard scripts:** Results must be copied to *TYPEReportREG.docx* in your working directory and recorded in your ARCHMOD\_EVAL.xlsx summary spreadsheet.

**Enhanced scripts:** The ModelTemplate.Rmd or *TYPE.Rmd* script is designed to document both your modeling procedures and results. It creates a report (.html format) in your working directory with the naming convention matching the name of the .Rmd file it's derived from but with an ending of *.html* (for example, SOILSITE\_V1\_PLAT.html). The report is divided into sections corresponding to the sections in your R script. You may review this report as you go along by clicking the Preview button at the top of the script window (Figure 10). Remember that you must first save the .RMD file for the report results to be updated. You must also record certain model statistics in the ARCHMOD\_EVAL.xlsx spreadsheet, but you extract this information from *MODTYPE\_VN\_REG.html* after your model is completed.

**Figure 10: Previewing R Modeling Results**



Note that when you open your .html model report, you will see a message box asking if you want to allow blocked content. Be sure to check the 'Allow blocked content' button. Otherwise, you will not see some of the results.

Internet Explorer restricted this webpage from running scripts or ActiveX controls.

Allow blocked content

## Saving Your Project

There are two options for closing a project.

- Select 'Close Project' from the File menu. This will close your project, but R will stay open and you can move on to a different project.
- Select 'Quit Session' from the File menu. This will end your R session and close R. The next time you open R, the last project you had opened will also open.

Either way, you will be prompted to save your project. Your project file will automatically save when R closes if you check the box next to 'Source on Save' box in the toolbar above the scripting window.

## Re-Opening a Saved Project

When you close your project and later re-open RStudio, your most recently used project will open. It is not necessary to re-run the entire script. However, there are several things you need to do in each session:

- Run the first chunk of code in the script to load all of the packages and libraries.
- Define the categorical variables using the [Categorical <c\(\)](#) function.
- Once you have completed these steps, you can proceed where you left off.

If you wish to open a different project from the most recent one you worked on, select Open Project from the File menu. Navigate to the appropriate working directory and select the \*.Rproj file.

## Overview of R Modeling Files

Both *TYPE*ModelTemplateVN.R and ModelTemplate.Rmd contain the baseline scripts for running the archeological site or survey predictive models. All scripts are organized into six sequential sections, as follows:

- Section 1: SETUP
- Section 2: DATA FORMATTING AND CLEANING
- Section 3: EDA (Exploratory Data Analysis)
- Section 4: PREDICTIVE MODEL TRAINING AND TESTING
- Section 5: RUN THE PREDICTIVE MODEL
- Section 6: EXPORT GEOREFERENCED PREDICTIONS TO CSV

These sections will be described in detail below.

## Section 1: SETUP

---

In this section, you will load the necessary R code for your work, set R parameters, load your data, and define your working directory.

## Load Packages and Libraries

R is modular software. Depending on which functions you need to use, you must load a set of packages and libraries that contain the code to run those functions. You must load these every time you start a new session.

The `pacman` package allows you to call libraries from packages not yet installed using the `p_load()` function, which will be used throughout the remaining R scripts. The 'car' library contains functions that will be used for data formatting. The 'psych' library contains functions for summarizing the sample data. Use the RStudio 'Help' tab to read more about each function and library.

**Standard Scripts:** In Section 1 of your script, the following lines of code load the packages and libraries needed for data formatting, exploratory data analysis (EDA), and modeling.

```
install.packages("pacman")
library(pacman)
p_load(car)
p_load(psych)
p_load(e1071)
p_load(MASS)
p_load(caret)
p_load(Rcmdr)
p_load(plyr)
p_load(Hmisc)
p_load(corrplot)
p_load(randomForest)
p_load(mlbench)
p_load(ROCR)
p_load(pROC)
p_load(OptimalCutpoints)
```

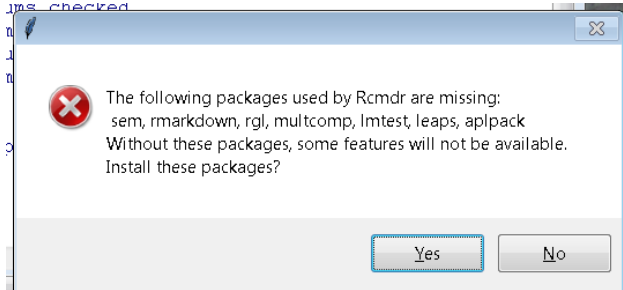
To load the packages, just select lines and click 'Run'. *This is the last time you will see instructions on how to run code.*

**Enhanced scripts:** The enhanced scripts load packages and libraries in the first chunk of code. You will notice that additional packages are required to run the .Rmd version of the models.

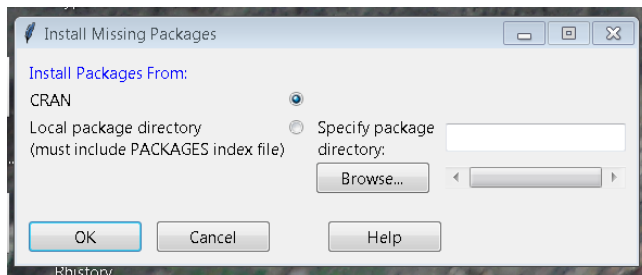
When you load Rcmdr, it will open an RCommander window. Simply close it. The package will run in the background without the interface being open.

## Select a CRAN Mirror

The first time you run RStudio, this warning may appear when installing (Rcmdr):



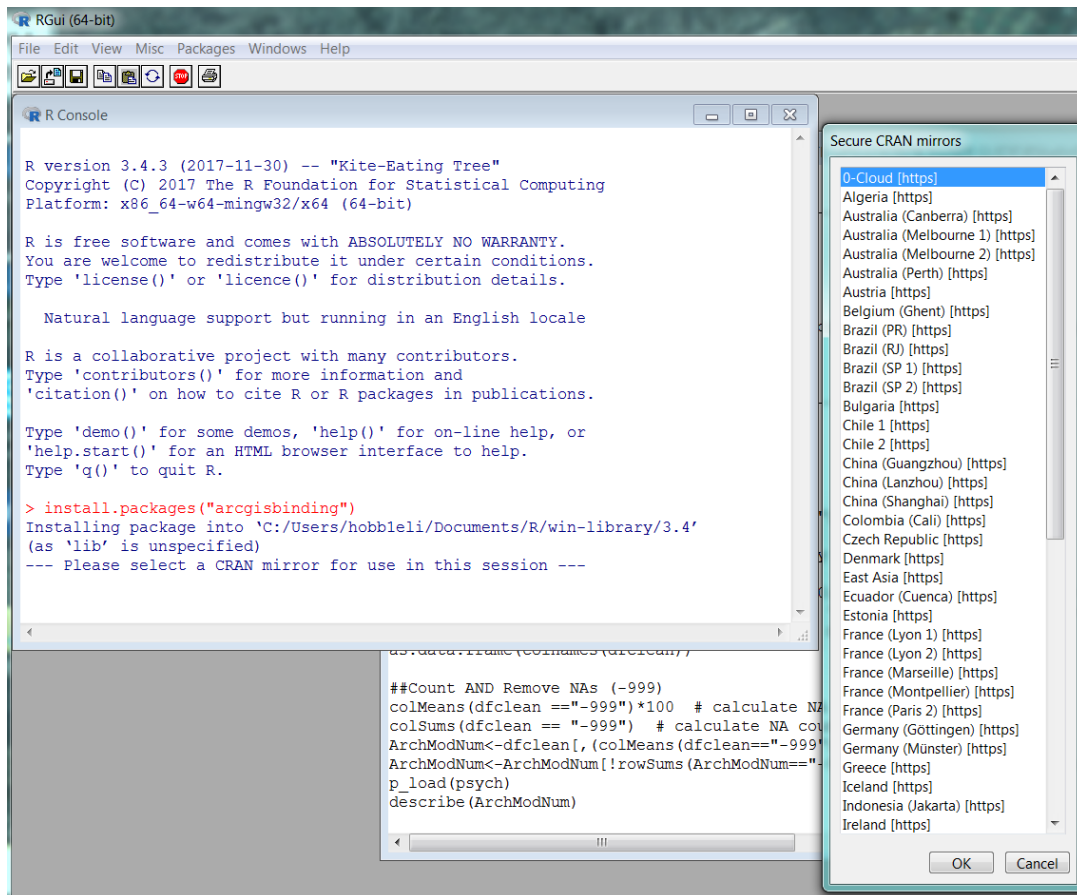
Clicking 'Yes' should produce this:



Clicking 'OK' loads the packages from CRAN (Figure 11). Both USA (Berkeley) and USA (IA) will work. You will not be prompted again once you select a CRAN mirror.

This applies no matter which version of the scripts you run.

Figure 11: Request to Install a CRAN Mirror



## Increase Maximum Number of Lines Printed

Depending on the number of records and variables in the model, some of the matrices produced exceed the default number of lines R will print. We have opted to increase this default.

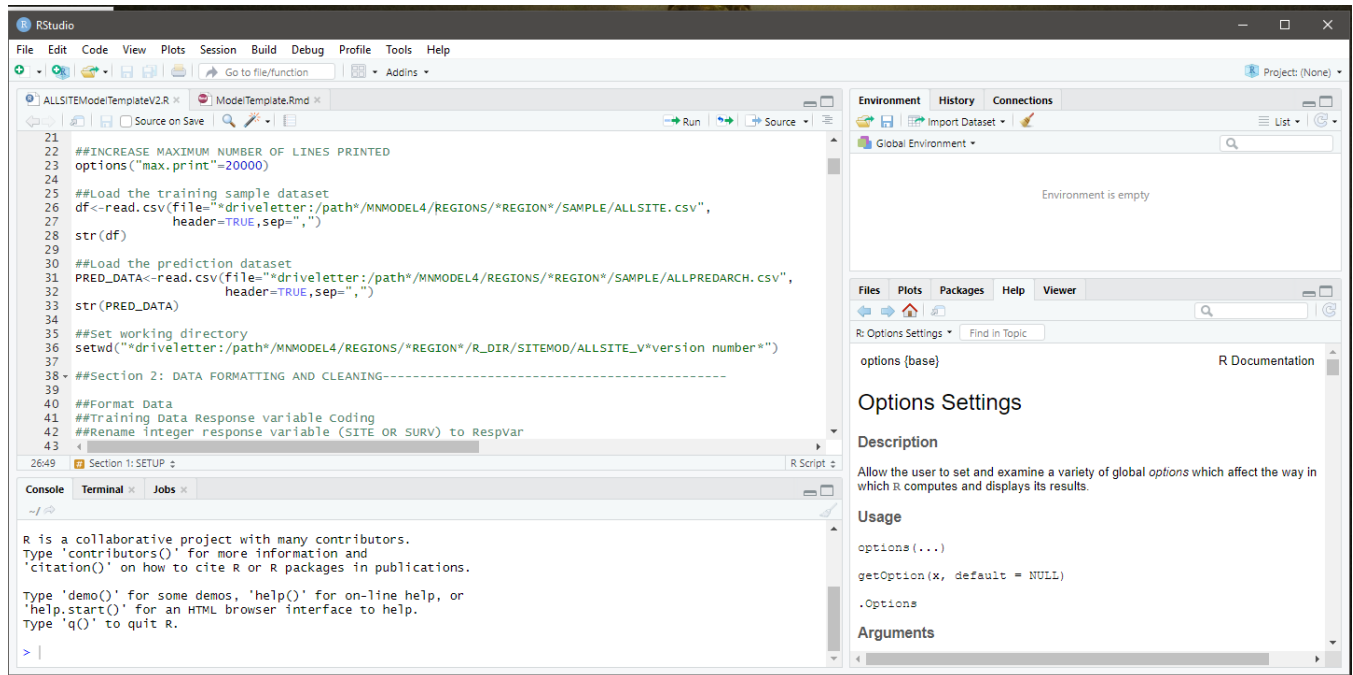
## Load the Data

**Standard Scripts:** You will load two sample datasets. The first sample dataset, called the training dataset, is for training the predictive model. It consists of the information sampled at site or survey points and background points, and it knows which points represent sites/surveys and which background. The second dataset, called the prediction dataset, is for generating predictions using the trained model. It contains the environmental data sampled with the 30 meter grid of prediction points, but no information about sites or surveys. Each type of model we run uses a comma delimited text file (.csv, the training data) as input ([Appendix B](#)). The text file locations are specified in your R file.

Figure 12 illustrates standard lines of code for loading data. You should see something very similar in your RStudio file. Lines 26 and 27 in Figure 12 read the training sample dataset .csv file into RStudio. Lines 31 and 32 load the predictive sample dataset into RStudio.



Figure 12: Loading Data



1. Edit your script to define the path to your training data.

Specify the correct path to the MNMODEL4 directory and the region you are working on. For example:

```
df<-read.csv(file="C:/MNMODEL4/REGIONS/BGWD/SAMPLE/ALLSITE1.csv",
```

Or:

```
df<-read.csv(file="z:/mndot_data/MNMODEL4/REGIONS/ANOK/SAMPLE/ALLSITE1.csv",
```

2. Edit your script to define the path to your prediction data.

Specify the correct path to the MNMODEL4 directory and the region you are working on. For example:

```
PRED_DATA<-read.csv(file="C:/MNMODEL4/REGIONS/BGWD/SAMPLE/ALLPREDARCH.csv",
```

Or:

```
PRED_DATA<-read.csv(file="z:/mndot_data/MNMODEL4/REGIONS/ANOK/SAMPLE/ALLPREDARCH.csv",
```

3. Edit your script to set the path to your working directory.

Specify the correct path to the MNMODEL4 directory and the region you are working on. For example:

```
setwd("C:/MNMODEL4/REGIONS/BGWD/R_DIR/SITEMOD/ALLSITE1_V1")
```

Or:

```
setwd("z:/mndot_data/MNMODEL4/REGIONS/ANOK/R_DIR/SITEMOD/ALLSITE1_V2")
```

#### 4. Load the training data.

- Select lines 26-28 of the R script.
- Click 'Run' at the top of the window.
- Notice that once the data are loaded, the `str(df)` command displays information about the data frame in the lower console window. This includes the number of rows ('obs.' for observations) and fields (variables) in addition to information related to each field. Verify that all fields are integer ('int').



Copy your results to your Model Report.



Record the number of sample points (rows) reported in your worksheet. You will need this number for future reference.



Calling '<-' means you are generating a new object in R. The new object will appear in the Global Environment window. The name of the new object (what it will be called in R) is to the left of '<-' . The expression to the right of '<-' defines how the new object will be created.

#### 5. Load the prediction data.

- Select lines 31-33 of the standard R script.
- Click 'Run' at the top of the window.
- The `str(PRED_DATA)` command displays information about the data frame in the lower console window. This includes the number of rows ('obs.' for observations) and fields (variables) in addition to information related to each field. Verify that all fields are integer ('int').



Copy your results to your Model Report. You will need to refer to them later.



Record the number of prediction points for your model in your worksheet. These will be used for a future calculation.



If the file is large, it might take a few minutes to load. You will know when the file is loaded when the R console prompts `>` followed by a blinking cursor.

- You have the option of viewing some of your results in a separate window (Figure 13). Double click the 'df' dataframe in the Global Environment window. This will open the dataframe in a new window. Quickly review the data to ensure that there are no anomalies and that the data loaded properly.

Note: Anomalies would include missing data fields, empty data fields, data fields with the invalid values, etc. We hope these have already been removed in ArcGIS.

**Figure 13: Optional Data View**

The screenshot shows the RStudio interface. The main window displays a data frame 'df' with 10,706 entries. The columns are: RespVar, UniqueID, LFORM, LSCAPE, WETSOIL, ASP\_RNG, CURV, ELEV, REL90, RGH, RGH90, SHELTER, SLOPE, TPI1000, TPI1MI, TPI250, TPI500. The console shows the command 'df <- read.csv()' and the output of 'colnames(df)' listing 23 variables: OBJECTID, POINT\_X, POINT\_Y, LFORM, LSCAPE, WETSOIL, ASP\_RNG, CURV, ELEV, REL90, RGH, RGH90, SHELTER, SLOPE, TPI1000, TPI1MI, TPI250, TPI500, TWI, RespVar, and RespVarCat.

- Set your working directory. This directory is where R will automatically save automated outputs such as pdf figures. Running `setwd()` function will set your working directory
  - Select line 36 of the R script.
  - Click 'Run' at the top of the window.

**Enhanced scripts:** ModelTemplate.Rmd determines the working directory and data paths in the second chunk of code, under the heading `##Set up region-specific parameters`. This code is smart enough to find your working directory (as this is where the script resides) and from that path it deduces the paths to your data files. If you

have not used MnModel 4 standard paths and file naming conventions as specified in earlier sections of this User Manual, this code will not work. The script does not actually load the data until the third and fourth chunks of code.



**Data Frames:** Notice that each imported .csv file is renamed (for the purpose of R) as 'df' or 'PRED\_DATA' and converted to a dataframe, which is a data matrix that recognizes data field formats such as text, numeric, integer, etc. You will see the dataframe ('df', for example) appear in the Global Environment window. Throughout the R script, as we remove variables, add variables, and make other edits to the data, those changes will occur in the dataframe, not in the .csv file. Also, you will notice that when some changes are made, a new dataframe name is specified (for example, 'df' becomes 'dfClean'). Be aware that if you start editing the script in any way, you need to be careful to specify the version of the dataframe that you want your changes to access. If you specify the wrong dataframe name (an earlier version, for example, before redundant variables were removed) you may not get the results you expected. The copies you made of the `str()` results will help you later if you have any questions about how your current dataframe differs.

## Section 2: DATA FORMATTING AND CLEANING

---

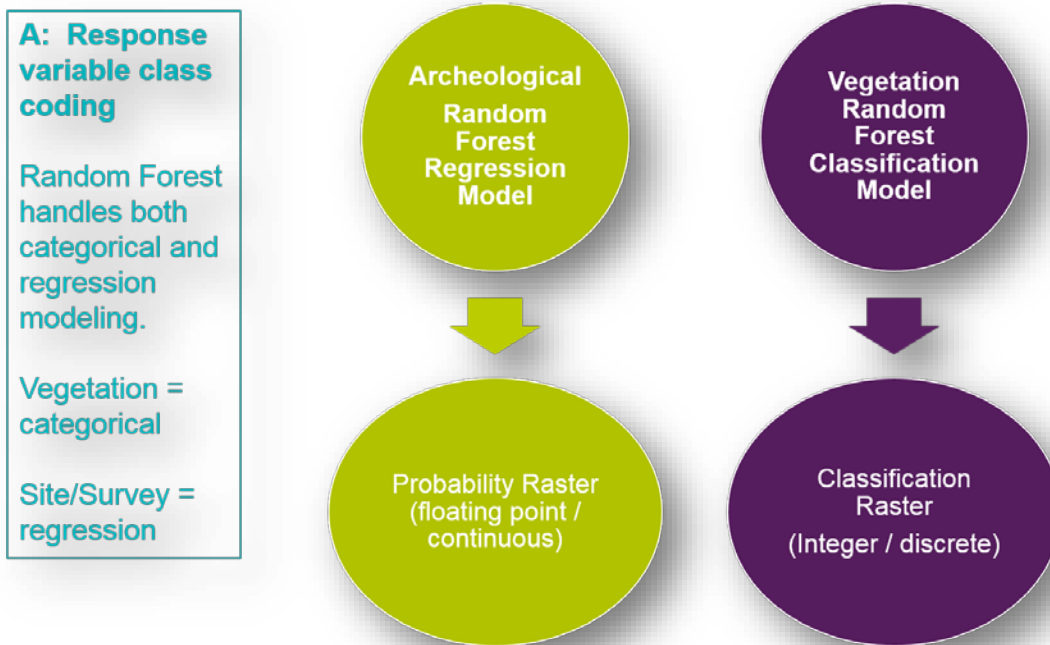
### Create Response Variables

Response variables are those variables that tell R which of your data records represent sites or surveys and which represent background points. The training datasets, ALLSITE1 and SOILSITE1, contain the variable SITE, which will have the values '1' for a site and '0' for a background point. The ALLSURV1 and SOILSURV1 datasets contain the variable SURV, which will have the values '1' for a survey and '0' for a background point. SITE and SURV are the 'response variables' for their respective models.

For the site and survey predictive models, we create a numeric response variable field ('RespVar') for completing EDA and to train and test the predictive model. We also create a 'dummy' classification, or categorical, response variable data field ('RespVarCat') to explore relationships between site and background locations and categorical predictor variables (i.e. landforms, landscape, and vegetation classes) **during the EDA process only**.

The numeric field type is important for predicting site or survey locations and communicates to R that 1) the site or survey predictive model should be regression (not classification) and 2) the predictive model outputs should be continuous or floating point. The Random Forest model, which is the recommended predictive model, runs both regression and classification (Figure 14).

Figure 14: Response variable coding tells R which Random Forest Predictive Model to run.



The prediction dataset is based simply on a grid of points and has no ‘knowledge’ of which are sites/surveys and which are non-sites/non-surveys. However, it is important that the prediction sample data contain the same data field names as the training sample data. This is because the trained model will need to recognize and account for all the variables in the prediction sample dataset that were used to train the model. Therefore, you must generate the same response variable fields (i.e. ‘RespVar’ and ‘RespVarCat’) as before but add them to the prediction data as ‘dummy fields’ with ‘0000’ placeholders.

**Standard scripts:**

1. Create response variables in training data

This line tells R to rename the training data field ‘SITE’ to ‘RespVar’:

```
colnames(df)[colnames(df)=="SITE"]<-"RespVar"
```

This line tells R to create a categorical variable from ‘RespVar’ and to call it ‘RespVarCat’:

```
df<-within(df, {RespVarCat <- Recode(RespVar, '1=1;0=0',as.factor.result=TRUE)})
```

Breaking down the syntax:

- The Recode() function references the ‘RespVar’ field in the ‘df’ dataframe to assign a ‘0’ value to background sample locations (RespVar = 0) and ‘1’ to site locations (RespVar = 1).

- The 'as.factor.result=TRUE' argument tells RStudio the 'RespVarCat' field being created is **categorical**.

The `as.data.frame(colnames())` function proves useful in many ways and will be called repeatedly throughout the R scripts. This function responds with the column number and name for each data field in your dataframe. At this point, we reference this information to verify that the response variables were correctly defined.

```
as.data.frame(colnames(df))
```



Copy your results to your Model Report.

## 2. Create 'dummy' response variables in predictive data

For the site or survey predictive model, you will use the following lines of code to add the dummy response variable data fields:

```
PRED_DATA["RespVar"]<-0000
```

```
PRED_DATA["RespVarCat"]<-0000
```

Use the `as.data.frame(colnames())` function again to verify that these variables were added to the prediction dataset.



Copy your results to your Model Report.

**Enhanced scripts:** The enhanced scripts perform exactly the same procedures, using the code chunk under the heading `###FORMAT DATA`.

## Remove Fields Not Necessary for Statistical Modeling

Table 2 generalizes the data fields that are necessary for building, training and running the R statistical models.

**Table 2: Data Fields Necessary for Modeling**

Variables	Sample Dataset
Predictor Variable	Training, Prediction
Response Variable	Training, Prediction
X,Y Coordinates	Prediction

To ensure that the correct variables are present for each version of the model, the predictor variables are sampled from one of two geodatabases (ALLARCHLIST.gdb or SOILARCHLIST.gdb). Sampling procedures are discussed in [Appendix B](#) of this Guide. The response variable for a site model is SITE and the response variable for a survey model is SURVEY. The training sample datasets do not require X,Y coordinate data fields. The prediction sample datasets, however, do require X,Y coordinate data fields so that each site or survey prediction is georeferenced for visualizing the predictions in ArcGIS. The variable naming conventions must be consistent, which is insured if both sets of points sample the same geodatabases. Depending on the variables selected for a given model, the list of variables for modeling and their order must be consistent between the training and prediction datasets. This requirement is also met if MnModel Phase 4 sampling procedures are followed ([Appendix B](#)).

There are several fields (columns) in our datasets that are holdovers from the GIS data and are not needed for modeling. You can remove columns by calling their position (i.e. column number) or their name. Both options are shown below. Most often, we will refer to variables by their names to avoid confusion.

**Standard scripts:** Remove unwanted variables from the training dataset by name:

1. Generate a list of variables to exclude, in this case the list is called 'droplist'. These are the variables that are not recognized as a response, predictor or X,Y coordinate variable.

Our standard list is:

```
droplist<-c("OID")
```

If your data differ from the standard, you may edit the list. For example:

```
droplist<-c("ObjectID","SURVCONF","UniqueID")
```

2. Produce a new dataframe called 'dfclean' that is void of the variables listed in the droplist.

```
dfclean<-df[,!(names(df) %in% droplist)]
```

3. Re-run the `as.data.frame(colnames())` function on the new 'dfclean' dataframe to make sure the variables were removed.

Breaking down the syntax:

- The `c()` function returns a vector or a list. Adding a negative (-) in front of the 'c' means we want to remove the list of variables.
- The `df[,!(names(df) %in% droplist)]` removes the dataframe fields listed in 'droplist'. We tell R to look at column names by including the syntax after the comma and use the logical negation `!` (meaning 'NOT').

Sometimes, it is easier to simply call the column number or position instead of its name. This is a personal preference and data management preference. For the prediction data, we use `as.data.frame(colnames())` to print the variable names along with their column position. Then we remove the 'OID' field from the predictor sample data by calling its position '1'. Note that the 'OID' field is removed from the training dataset by name. We would add any variables to this list congruent with the variables we removed from the training dataset. You can separate different column numbers by a comma (e.g. `-c(1,3,6)`) or if there are consecutive variables you can separate by a colon (e.g. `-c(1:4,6)`, which will remove columns 1 through 4 and column 6 in the dataframe).

```
as.data.frame(colnames(PRED_DATA))
```

```
PRED_DATA<-PRED_DATA[-c(1)]
```

Notice that in the previous line of code we did not create a new dataframe. Re-run the `as.data.frame(colnames())` function on the PRED\_DATA dataframe to make sure ObjectID was removed.

**Enhanced scripts:** These run the standard code as described above. If your data are not standard, you must edit the code accordingly.

## Calculating NULL Frequencies and Removing NULL Values

Statistical procedures do not know how to handle NULL values. The only valid NULLs in our data should all have been recoded to '-999'. We will refer to these as NULLs. This subsection describes how to calculate NULL frequencies and remove NULL values in RStudio. The R functions `colMeans()` and `colSums()` will display the NULL frequency and NULL counts for each predictor and response variable in the 'dfclean' and 'PRED\_DATA' dataframes.

Our data preparation steps ([Appendix B](#)) discussed finding and eliminating NULL values in the GIS data. Soils data have valid NULL values where soil scientists did not collect data. Any NULL values should have been caught when the GIS data were being prepared for modeling and calculated to '-999'. Most NULL values in the soils data should be concentrated in the areas identified in MODMASKPL and should have been removed from the datasets for SOILSITE and SOILSURV models. The variable WETSOIL may have NULL values in the ALLSITE and ALLSURV datasets. If this is the case, the procedures below will remove the variable. The number of NULL values in WETSOIL will be greatest in the NE MN counties that largely lack soils data.



NULL values are not necessarily congruent between data fields. In other words, we are likely to have valid values for terrain variables in the same record or row as NULL values for one or more soil variables. Removing all records containing NULL values might result in a potentially large number of data records being removed for certain types of analyses. To avoid this, it is helpful to first identify data fields with high NULL frequencies and remove these fields from the dataframe instead of the records.



R will display NULL values not set to '-999' as 'NA'. If you see these 'NA' values in any fields, in any of your output from R, you need to go back to the GIS and figure out where they came from, as they should not be there.

### Standard Scripts:

#### 1. Calculate NULL frequencies in training and predictive data

The first step is to determine whether, and where, you have NULL values. The `colMeans()` function will calculate frequencies, while the `colSums(dfclean == "-999")` function will provide counts. At this point, frequencies are most important.

```
colMeans(dfclean == "-999")*100  
colMeans(PRED_DATA == "-999")*100
```



Copy these results to your Model Report.



Record the names of any variables with >5 percent NULL values in your spreadsheet.

#### 2. Review Variables with NULL Values

If you found NULL values in anything except soil variables, or if you found a very high frequency of NULL values in any variable, you may need to break out of R and examine variables in ArcGIS to see if the NULL values are valid. We expect NULL values only in soil variables. Distance variables and variables derived from terrain data or from the Landscape Model should not have NULL or 'NA' values. If you find that they do, you must determine the cause and correct it.

If variables contain >5 percent NULL values, we will want to remove the variable from the model. If running the ALLSITE or ALLSURV model and WETSOIL contains any NULL values, we will want to remove the variable. If any variable besides WETSOIL (or WETSOIL in the SOILSITE or SOILSURV model) contains less than 5 percent NULL values, we will remove only the records with NULL values and leave the variable.

#### 3. Remove Variables with NA Values

The actions you take at this point depend on the frequencies of NULL values in your sample data.

- a) No NULL values in any variable.

It may be that NO variable in either dataset has a NULL value. This is most likely to happen in the ALLSITE or ALLSURV models where WETSOIL is the only soils variable. If this is the case, you run the lines:

```
ArchModNum<-dfclean  
PRED_NUM<-PRED_DATA
```

These lines of code will rename the current dataframes so that they are consistent with the dataframe names going forward in the script. This will not change the number of records in your dataset.



The `psych::describe()` function will print a nice description of your data. You may use this to verify that no variable has a minimum value of '-999'.

- b) The variable SHELTER has NULL values

'-999' is a valid value for the SHELTER variable. If NULL values are reported in SHELTER, ignore them. If this is the only variable with NULL values, use the code specified above to rename your files. This will not change the number of records in your dataset.

- c) ALLSITE or ALLSURV model and WETSOIL has NULL values.

If WETSOIL contains even a single NULL value, the variable must be removed. This is to avoid creating 'holes' in the GIS model developed from the prediction points. Such holes are allowed only in the SOILSITE and SOILSURV models. The purpose of the ALLSITE and ALLSURV models is to provide predictions that fill in these holes.

If WETSOIL has ONE OR MORE NULL values in either dataframe, it must be removed from both datasets. These lines of code will create a list of variables to remove from your data, then remove those variables.

```
RmVariableList<-c("WETSOIL")  
ArchModNum<-dfclean[,!(names(dfclean) %in% RmVariableList)]  
colSums(ArchModNum == "-999")  
PRED_NUM<-PRED_DATA[,!(names(PRED_DATA) %in% RmVariableList)]  
colSums(PRED_NUM == "-999")
```

You run the `colSums()` function for each revised dataframe to verify that there are no longer any NA values present.

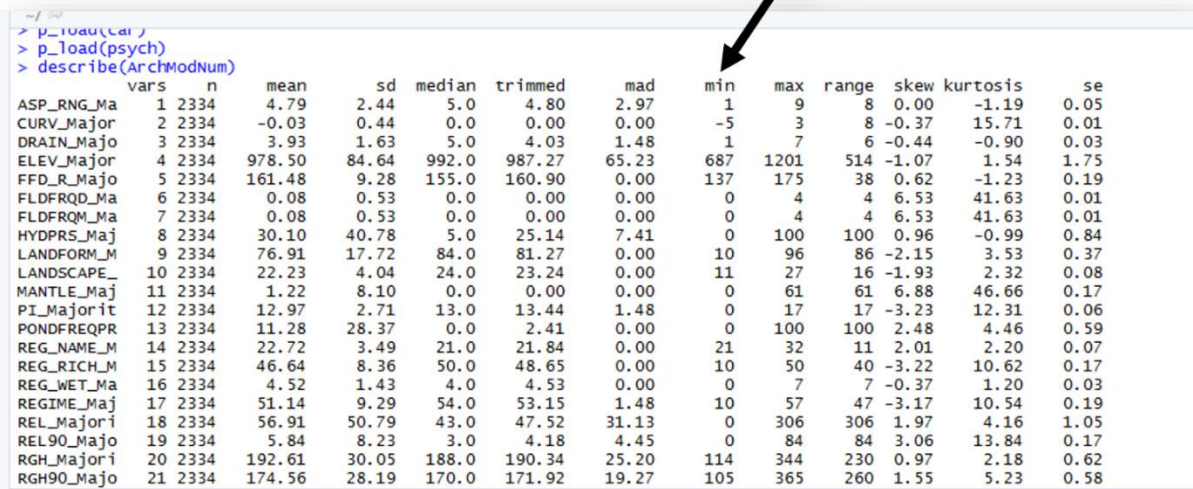


Copy the `colSums()` results to your model report. Optionally, you can also run the `psych::describe()` results for the two data frames. These should verify that no remaining variable has a minimum value of '-999' (Figure 15).



If you do remove WETSOIL, record this information in your worksheet. Removing a variable will not change the number of records in your dataset.

Figure 15: Descriptive Statistics



- d) If you are running SOILSITE or SOILSURV, any soil variable may have NULL values.
- i. If the frequency of a variable in EITHER dataset is greater than five percent, the variable will need to be removed from BOTH datasets. You will create a list of variables to remove (RmVariableList), add one or more variables to the list, and remove the variables from both the training and the predictive data. For example:

```
RmVariableList<-c("CACO3","WETSOIL")
ArchModNum<-dfclean[,!(names(dfclean) %in% RmVariableList)]
PRED_NUM<-PRED_DATA[,!(names(PRED_DATA) %in% RmVariableList)]
```



Record all variables you removed in your worksheet.



Record your colSums() results, which should confirm that your NULL frequency is now '0' for all variables.

- ii. Whether or not you found variables with greater than five percent frequency of NULLs, there will undoubtedly be isolated records here and there with NULL values for soils data. Since the

number of these records will be relatively small, and since those locations will be modeled by the ALLSITE and ALLSURV models, we can safely delete these rows without removing the variables. These lines will remove any remaining rows of data that contain NULL values:

```
ArchModNum<-dfclean[!rowSums(dfclean=="-999"),]  
colSums(ArchModNum == "-999")  
PRED_NUM<-PRED_DATA[!rowSums(PRED_DATA=="-999"),]  
colSums(PRED_NUM == "-999")
```



Record your `colSums()` results, which should confirm that your NULL frequency is now '0' for all variables.



Since you run this code whether or not you removed variables, the number of records in your dataset will have changed. Run `psych::describe()` to determine the new values and record them in your worksheet.

```
psych::describe(ArchModNum)  
psych::describe(PRED_NUM)
```

**Enhanced scripts:** The enhanced scripts count the NULLs and automate the decisions and steps described above.



**RmVariableList:** This list of variables to be removed from the datasets will be used throughout the data formatting, EDA and modeling procedures. Variables may be removed because they contain too many NULL values, because they are strongly correlated with other variables, or for other reasons. You must keep track of the most recent version of this list and add to the variables as more are identified for removal. Throughout EDA, these variables will be removed only from the training data. Thus it is critical that when you apply the list to the predictive data, in Section 5 of your script, you use the most current version of the list so that it removes exactly the same variables.

## Factor Counts

'Factor' is R's term for a categorical variable. 'Factor levels' refers to the total number of unique values in any given categorical variable. The only factors in our data are LFORM, LSCAPE, and VEGMOD. The number of 'levels' or unique values in any of these factors will vary from one region to the next and, because of the difference in sampling patterns and density, between the training and predictive data. For R to be happy with our categorical data, the following must be true:

- Ideally, there should be no 'factor level' with fewer than 15 records in the training data. Occasionally it will work to have as few as eight records. However, the training data will be split into two sets (a 75 percent/25 percent split) for training and testing. If all of the records fall into the training sub-dataset (the 75 percent portion) and none are in the testing data, there will not be a problem. However, if they

are all in the testing data and none into the training data, R will produce an error. In short, there must not be any 'factor levels' in the testing data that were not used to train the model. The easiest way to make sure this happens is to have a minimum of 15 records in each factor level of the original training data.

- There MUST be the same number of 'factor levels' in the training and predictive datasets. Moreover, these must be exactly the same factor levels (i.e. the column of values must match). When it is time to apply the model developed using the training data to the predictive points, the two datasets must match or the model will not run.

In reality, the training dataset frequently has factor levels with fewer than 15 values. Sometimes the predictive data has factor levels with fewer than 15 values. The predictive data almost always has more factor levels than the training data. This is why we do factor counts and re-assign values to factor levels to combine them with less rare, but similar, data.

Determining which values to use will depend on the data. For example, if the rare landform is '44' (Floodplain, Developing Features), it can be combined with landform '45' (Floodplain, Featureless) or '49' (Floodplain, Undifferentiated). Some of these choices will not be so obvious. The user must have a basic understanding of the categorical data and the relationships between classes. If necessary, consult ArcMap to determine what other classes are in close proximity or similar topographic positions to the rare value. Be consistent about value substitutions. Keep track of the substitutions you use and refer back to your list when making decisions for subsequent models.



**Class Imbalance:** 'Class Imbalance' refers to the situation in a categorical variable where some classes have very high frequencies and others have very low frequencies. In general, categorical variables with a high number of levels tend to exhibit imbalance, as some categories are represented by large numbers of records while other categories are rare. Not only can extreme imbalance prevent the model from running, but classes with very low counts and classes with very high counts offer little information for predictive modeling. This may result in the predictor variable holding less overall importance.

One way to reduce imbalance is to group smaller classes with larger classes. We must do this for the smallest classes to ensure that the model will run. If the classification system is hierarchical, as are the MnModel vegetation and geomorphology data, another option is to use only the higher levels of the hierarchy for modeling (for example using only LSCAPE and not LFORM). There may be other ways to address the issue of imbalanced categorical variables as well, such as determining which categories are important for prediction and creating distance variables from them.

**Standard Scripts:** This is a complicated procedure and requires more 'hands-on' work than any other part of the modeling process. It is impossible to fully automate. The enhanced script requires nearly the same level of user interaction.

## 1. Count your factors

The `count()` function will provide you with a count of the number of records in each factor level of a categorical variable.

```
count(ArchModNum, 'LFORM')  
count(PRED_NUM, 'LFORM')
```

The 'freq' column indicates the counts for each class listed under the factor value column (in this example, 'LSCAPE'). Note the classes are listed by their corresponding valid value. The counts represent both site and background points. The counts for the '15' (Dissected Bedrock Uplands), '24' (Stagnant Ice) and '26' (Valley Margin) classifications are orders of magnitude higher in comparison to other classifications for this predictor variable, especially classes identified as '10' (Active Ice), '13' (Collapsed Outwash Plain), '16' (Eolian), and '25' (Tributary Fan). At this point, we are concerned with the ones with fewer than 15 occurrences. We will consider the remaining imbalances in future modeling phases.

```
> count(Vegc1ean, 'LSCAPE')  
  LSCAPE freq  
1      10   13  
2      11  191  
3      13   36  
4      15 5135  
5      16    1  
6      17  971  
7      18  607  
8      24 2588  
9      25   58  
10     26 2442  
11     27  452  
> |
```



Each time you perform a count on a variable, copy the results into the appropriate table in your model report. You will perform counts of each categorical variable once to determine where the problems are and again, after editing the data, to verify that the problems have been resolved.



Problems to look for (either of this may prevent the models from running):

- If the frequency of any category in the training data falls below about eight.
- If you have a different number of factor levels in the two sets. This means that the prediction points sampled a category that was not sampled by the training data (site/survey or background points)

In Table 3, the LFORM variable has 32 levels in the training data and 37 levels in the prediction data. That is because LFORM values 22, 58, 83, 92, and 97 are missing from the training data. Also, the training data has fewer than 15 records of ten other factor levels.

**Table 3: Initial Counts of LFORM**

TRAINING DATA			PREDICTION DATA		
<code>&gt; count (ArchModNum, ' LFORM' )</code>			<code>&gt; count (PRED_NUM, ' LFORM' )</code>		
	LFORM	freq		LFORM	freq
1	10	35	1	10	31751
2	15	68	2	15	82382
3	17	1	3	17	903
4	20	120	4	20	106981
5	29	758	5	22	946
6	37	8	6	29	766768
7	38	2	7	37	9640
8	40	11	8	38	3190
9	42	153	9	40	18964
10	43	128	10	42	27376
11	44	4	11	43	159649
12	45	57	12	44	3902
13	47	2	13	45	66227
14	48	39	14	47	7317
15	49	103	15	48	52900
16	52	19	16	49	101739
17	53	32	17	52	11793
18	56	276	18	53	20467
19	59	1	19	56	251571
20	62	5	20	58	568
21	63	51	21	59	738
22	65	249	22	62	2214
23	68	383	23	63	80497
24	72	93	24	65	407331
25	73	17	25	68	259043
26	77	7	26	72	98251
27	78	4	27	73	14050
28	80	22	28	77	7172
29	84	7546	29	78	6829
30	87	11	30	80	20532
31	93	332	31	83	613
32	96	169	32	84	6828486
			33	87	21045
			34	92	79
			35	93	282342
			36	96	191122
			37	97	12

Print the page with this table from your Model Report and do the following:

- Scan the list on the left (values in ArchModNum dataframe) and highlight any with 'freq' less than 15.
- Highlight the values in the same LFORM categories in the list on the right (PRED\_NUM dataframe), using the same color highlighter.
- Now scan the list on the right and look for any LFORM values that are not in the list on the left. Highlight these in a different color.

## 2. Edit Your Data

To resolve the issues, you need to re-assign values of rare variable categories to combine them with less rare categories. For example, if the rare landform is '44' (Floodplain, Developing Features), it can be combined with landform '45' (Floodplain, Featureless) or '49' (Floodplain, Undifferentiated).



### Caveats:

- The value that you assign to the rare feature MUST already be in the training dataset. It may be one of the rare values if combining with other rare values produces 15 or more records.
- Values that are in the predictive data and not in the training data must be re-assigned to values that are in the training data.
- Any values edited must be edited in both datasets if they occur in both originally, but are edited only in the predictive data if they are originally missing from the training data.
- In the end, the training data and the predictive data must have the same number of 'factor levels' (unique values for a categorical variable) and these values must match.
- If you make a mistake in coding these re-assignments and put something into the wrong category, you need to start a new project and run the script from the beginning.
- We need to be consistent about which values we substitute. Keep track of substitutions you have used and refer to them when developing subsequent models. Try to keep substitutions as consistent as possible.

You must edit the next lines of code to implement reclassification of the rare factors. Use the highlighted lists you created to:

- Find LFORM values you want to replace. These will go on the left side of the expression.
- Verify that the replacement value on the right side of the expression is in your training data.
- Copy and edit the basic lines of code for each rare value in each dataset.

```
ArchModNum$LFORM[ArchModNum$LFORM==*Original Value*]<-*New Value*
```

```
PRED_NUM$LFORM[PRED_NUM$LFORM==*Original Value*]<-*New Value*
```

For example:

```
> PRED_NUM$LFORM[ PRED_NUM$LFORM==17 ] <- 68  
> PRED_NUM$LFORM[ PRED_NUM$LFORM==22 ] <- 49  
> PRED_NUM$LFORM[ PRED_NUM$LFORM==37 ] <- 93  
> ArchModNum$LFORM[ ArchModNum$LFORM==17 ] <- 68  
> ArchModNum$LFORM[ ArchModNum$LFORM==37 ] <- 93
```



Note that you will be recoding more values in the predictive data than in the training data.

- Be sure to edit the replacement codes for both the predictive and training data.
- Be sure to select the same replacement values (right side of equation) in both lists.



Copy just the lines that you ran into your Model Report so that you have an exact record. This is any easy place to make mistakes.





Finally, you will run and record your `count()` functions again to verify that the corrections had the desired effect. The results of applying these corrections on an example from the Vegetation Model are displayed in Table 4.

**Table 4: Corrected Counts of LSCAPE**

TRAINING DATA			PREDICTION DATA		
<code>&gt; count(Vegc1ean, 'LSCAPE')</code>			<code>&gt; count(PREDVeg_DATA, 'LSCAPE')</code>		
LSCAPE	freq		LSCAPE	freq	
1	11	191	1	11	184680
2	13	36	2	13	36202
3	15	5135	3	15	4880030
4	17	971	4	17	985952
5	18	607	5	18	543543
6	24	2601	6	24	2454539
7	25	58	7	25	56199
8	26	2442	8	26	2211155
9	27	453	9	27	455083

**Enhanced Scripts:** This is the first section of the enhanced modeling script that requires user input to properly complete. This procedure cannot be automated. The enhanced scripts display the factor counts differently than the standard scripts, and editing procedures differ as well. You will do your editing in the variables.R script (see third step below), not in the master script.

1. First, rename the dataframes by running the chunk illustrated below. This chunk of code immediately precedes the script's heading for the Factor Counts.

```

342
343 > ##simply rename your dataframes so that they will be consistent with
    the rest of the script
344
345 If you accidentally mess something up while doing factor
    substitution, rerun this chunk to reset PRED_NUM back to its original
    setting.
346 > ```{r rename-dataframes}
347
348 ArchModNum<-dfclean
349 PRED_NUM<-PRED_DATA
350
351 # psych::describe(ArchModNum)
352 # psych::describe(PRED_NUM)
353

```



Factor Counts are the first part of the modeling process requiring user input. Because this chunk is just above the code for Factor Counts, the user may click on the 'run previous' icon (the dark grey down arrow with a green bar beneath) to run all previous steps. All of the chunks before this one have been automated and do not require user input to run properly. If this icon is used to run all previous

portions of the script, the chunk itself must be run next by clicking the green arrow icon in the upper right corner of the chunk to establish the factor level data.

2. This chunk may be rerun throughout the factor level substitution process to refresh the factor level data and ensure that factor level replacements are happening in one chunk run for each of the factors. This allows for proper generation of a summary report after the user edits the variables.R script (see below). Run the chunk illustrated below to determine and display factor level values for the LFORM variable and their corresponding counts. This will help the user make informed decisions about the consolidation of factor levels.

```

354
355 - ## FACTOR COUNTS
356 - ## Search for and correct categorical variable values with too few
      (<15) records
357
358 - ### Edit values of rare landforms in TRAINING data
359
360 - ```{r refactor-lform-values}
361   source('variables.R')
362
363   updated_data <- remap_factor_values(ArchModNum, PRED_NUM, "LFORM",
      lform_f_map, lform_t_map)
364   ArchModNum <- updated_data[['0']] # Re-assign ArchModNum to reflect
      remapped values.
365   PRED_NUM <- updated_data[['1']] # Re-assign PRED_NUM to reflect
      remapped values.
366   updated_data[['table']] # Display table showing results
367   ```

```

Running this chunk will display information about LFORM factor levels. This table shows each variable value number and the frequency or count of records corresponding to that value in both the sample and predictive datasets.

SAMPLE_LFORM	SAMPLE_FREQ	PRED_LFORM	PRED_FREQ
1	17	2	17
2	29	97	29
3	30	1	30
4	35	12	35
5	36	161	36
6	42	12	38
7	43	2	42
8	44	1	43
9	45	3	44
10	49	23	45



When displaying this table, it is best to show only 10 entries at a time. When more than 10 entries are specified, some factor levels present in the data may fail to display. Moreover, the option to scroll through factor levels is available only when the number of factor levels in the sample and predictive factors is identical. These display issues could cause factor substitution errors if the modeler is unaware of the missing values.

As with the standard scripts, there should be no factor level with fewer than 15 records in the sample data, although fewer may work (though you are taking a chance). There MUST be the same number of factor levels in the training and predictive datasets.

3. To consolidate factor levels, first determine which sample values have fewer than 10 records and record these in the corresponding 'f\_map' lines in variables.R. These are the 'from' values, the ones that will be changed.

```
32 # Local variables you will change as you go through the modeling process.
33
34 lform_f_map <- c(17, 30, 43, 44, 45, 53, 56, 59, 66, 68, 77, 87, 97)
35 lform_t_map <- c()
36
```

Notice that the numbers 17, 30, 43, 44, and 45 in the 'lform\_f\_map' line in the variables.R example correspond to factors with frequencies fewer than 10 in the table in step 2 above. These and all the other factor levels with fewer than 10 records must be combined with each other or with other factor levels to avoid modeling problems.

4. Use the 'lform\_t\_map' line of the variables.R script to list the new values for each of the rare factor levels. Be sure to list the new values in the same order as the original values to which they correspond.

```
32 # Local variables you will change as you go through the modeling process.
33
34 lform_f_map <- c(17, 30, 43, 44, 45, 53, 56, 59, 66, 68, 77, 87, 97)
35 lform_t_map <- c(65)
36
```

In the example above, the factor level 17, 'Beach', with two records, is being combined with 65, 'Lake Basin', with 73 records.



The value that you assign to the rare feature MUST already be in the training dataset. It may be one of the rare values if combining rare values produces 10 or more records. Try to keep your substitution decisions consistent from one model to the next.

When substitutes for all of the factor levels with fewer than 10 records have identified, the corresponding 'f' and 't\_map' lines in variables.R should look much like the image below.

```
32 # Local variables you will change as you go through the modeling process.
33
34 lform_f_map <- c(17, 30, 43, 44, 45, 53, 56, 59, 66, 68, 77, 87, 97)
35 lform_t_map <- c(65, 29, 93, 49, 49, 84, 29, 93, 49, 29, 84, 49, 93)
36
```

5. After coding the rare factor values and their substitutions in variables.R, perform the following steps:
  - a. Save the updated variables.R file.
  - b. Re-run the 'r rename-dataframes' chunk that precedes the factor counts code (see step 1 above).
  - c. Re-run the chunk that creates and displays the table of factor levels and their frequencies (step 2 above).

	SAMPLE_LFORM	SAMPLE_FREQ	PRED_LFORM	PRED_FREQ
1	29	104	29	948319
2	35	12	35	168825
3	36	161	36	1426290
4	42	12	38	2481
5	49	35	42	21376
6	57	10	49	179042
7	65	75	52	84
8	80	16	54	148
9	83	25	57	62860
10	84	1262	61	13

Showing 1 to 10 of 22 entries

Previous 1 2 3 Next

Step 5c should produce a table showing that all factor levels with fewer than 10 sample records have been consolidated into factor levels that have 10 or more sample records.

6. Next you must ensure that the factor levels are identical for both the sample and predictive dataset. In the table above you can see that factor level 38 is present in the predictive data, but not in the sample data.

In the variables.R file, list all of the factor levels in the prediction factor variable not present in the sample factor variable. Add these to the end of the list you created in the 'f\_map' line (step 3 above). You'll likely have to cycle between pages of the table to do this effectively.

```

32 # Local variables you will change as you go through the modeling process.
33
34 lform_f_map <- c(17, 30, 43, 44, 45, 53, 56, 59, 66, 68, 77, 87, 97, 38, 52, 54, 61, 62, 67, 69, 71, 73)|
35 lform_t_map <- c(65, 29, 93, 49, 49, 84, 29, 93, 49, 29, 84, 49, 93)
36

```

Repeat the process described in step 4 above to add factor substitution values to the 't\_map' line of your variables.R script.

```

32 # Local variables you will change as you go through the modeling process.
33
34 lform_f_map <- c(17, 30, 43, 44, 45, 53, 56, 59, 66, 68, 77, 87, 97, 38, 52, 54, 61, 62, 67, 69, 71, 73)
35 lform_t_map <- c(65, 29, 93, 49, 49, 84, 29, 93, 49, 29, 84, 49, 93, 84, 84, 93, 83, 84, 65, 49, 29, 49)
36

```

- After completing the 'f' and 't\_map' lines, checked the results by repeating step 5 above. Review the new table related to ensure that the sample and predictive datasets have identical factor levels.

	SAMPLE_LFORM	SAMPLE_FREQ	PRED_LFORM	PRED_FREQ
1	29	104	29	955692
2	35	12	35	168825
3	36	161	36	1426290
4	42	12	42	21376
5	49	35	49	179164
6	57	10	57	62860
7	65	75	65	769540

If your two datasets are ready for modeling, sample and predictive data will have the same factor level values on the same rows. Both the two sample columns and the two predictive columns should have exactly the same number of rows.

- You will repeat steps 2-7 three times, once for each of the categorical variables present in the model. These variables are LFORM, LSCAPE, and VEGMOD.



If you make a mistake during factor level consolidations, no worries, you can refresh the process by running the 'r rename-dataframes' chunk (step 1).

## Response Variable Counts

The last thing you will do prior to exploratory data analysis is to determine whether you have enough sites or surveys to proceed with modeling. Where we have large areas lacking soils data, we might run into very low site in SOILSITE1 or SOILSURV2 models. The situation is less likely in ALLSITE1 and ALLSURV1 models, but not impossible. The minimum number of sites or surveys for the random forest model should be about 100. However, we have successfully run models with as few as 40 sites.



Both script versions use the same `count()` function to count RespVar. Remember, you have RespVar only in the training data.



Record the number of site/survey points and the number of background points in your worksheet.

If it appears you have too few sites or surveys for modeling, your only option is to combine the region with an adjacent, similar region for modeling. Contact the Research Director if you find yourself in this situation.

## Generating a Call List for Factors

Now that we have balanced the factor levels for our categorical variables, we must tell R that they are factors. This will allow us to use these factors in future analyses. Specifying the response variable as 'factor' tells R to run a classification random forest model, which is the case for the vegetation model (Landrum and Hobbs 2019). Specifying the response variable as 'numeric' will tell R to run a regression random forest model, which is shown in the next Section of the R script. The Phase 4 categorical variables are LFORM, LSCAPE, and VEGMOD. We also have an alternative response variable, RespVarCat, that we will define as categorical. 'RespVarCat' is used only for EDA.

### Standard Scripts:

1. Use the `Categorical<-c()` function to create a list of categorical variables.

```
Categorical<-c("LFORM","LSCAPE","VEGMOD","RespVarCat")
```

2. `ArchModCat<-ArchModNum` generates a new dataframe called 'ArchModCat'
3. `ArchModCat[,Categorical]<-data.frame(apply(ArchModNum[Categorical],2,as.factor))` calls the categorical list and converts the fields listed from numeric to categorical or factor fields.
4. `str(ArchModCat)` is a QAQC step to ensure the correct data fields were converted.



`str(ArchModCat)` will display the variable names and types, which you should copy to your Model Report. The categorical variables should read as 'Factor' and list associated levels (i.e. valid values).

### Enhanced Scripts:

The enhanced scripts use the same R functions to produce the list of factors.



In the R scripts we generate two dataframes called 'ArchModNum' and 'ArchModCat'. The dataframes contain the same information EXCEPT that the 'ArchModNum' treats categorical variable class types as numeric and the 'ArchModCat' recognizes categorical class types. Why? Because some of the exploratory data analysis (EDA) functions, introduced in Section 3, cannot use categorical variable class types but are still useful for understanding patterns in categorical data and their relationships with numeric variable class types. Consequently, we keep these categorical variables in integer or numeric format to run certain EDA functions. This is for EDA only. The remainder of the R scripts must use the 'ArchModCat' dataframe, which recognizes categorical class types. If you use the 'ArchModNum' dataframe you will generate an inappropriate predictive model.

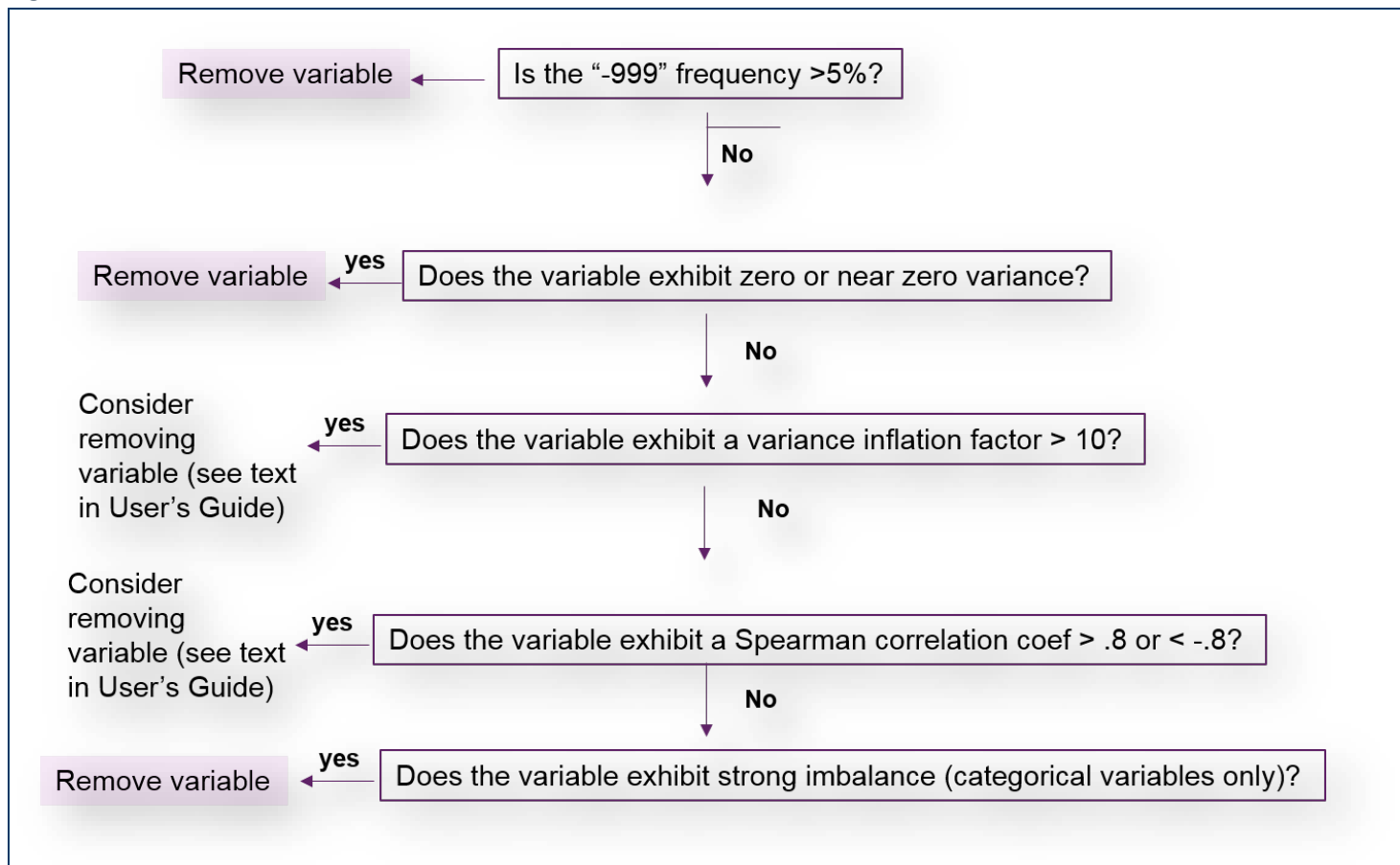
## Section 3: EDA (Exploratory Data Analysis)

Exploratory Data Analysis (EDA) uses only the training sample data. It is not possible to analyze trends in the prediction data because that dataset lacks the response variable. Thus the trends identified pertain only to a subset of the real world. If you change the composition of the sample dataset, by adding or removing either sites or background points, you can change the results of the EDA.

### Statistical Decision Matrix

The generalized decision matrix (Figure 16) will help users complete the exploratory data analysis for archeological predictive modeling. The first step, removing variables with more than five percent NULL values, has already been done.

Figure 16: Statistical Decision Matrix



### Summary Statistics (Site Vs Background)

The first step of EDA is to generate summary statistics using the `describeBy()` function. This code is the same in both versions of the scripts.

```
psych::describeBy(ArchModCat, ArchModCat$RespVarCat)
```

- Since we've now installed two libraries that share this function, it is necessary to tell RStudio which library to reference. We call the psych library by using 'psych::' argument.
- We also would like to compare summary statistics between site and background locations, so we group the statistics using the `describeBy()` function instead of `describe()`, then call for the dataframe ('ArchModCat') and the variable in the dataframe that identifies the groupings ('ArchModCat\$RespVarCat').



Background statistics for the two response categories (site/survey vs. background) in the 'ArchModCat' dataframe will be listed in the R Console first, followed by their statistics. If running the **standard scripts**, you will need to cut and paste these tables into your Model Report.

It can be helpful to compare mean and standard deviation values between groups. It is also helpful to study the skewness and kurtosis of the data. The following subsections breakdown these summary statistics into more practical terms.



If the `describeBy()` function does not display statistics by groups, make sure you ran all the lines of code in Section 2 above. In this case, it is likely the `RespVarCat` data column was not generated.

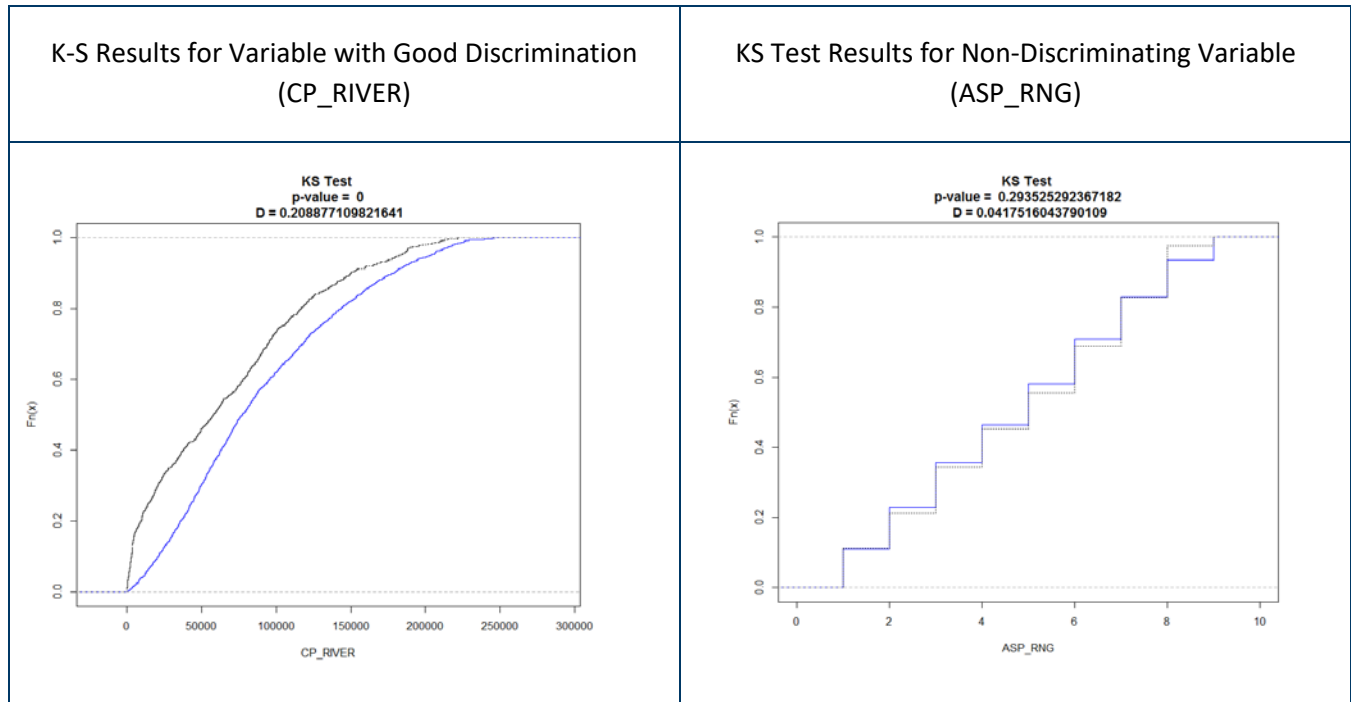
### Kolmogorov-Smirnov Test (K-S Test)

The purpose of comparing the mean and standard deviation between groups is to see if the environmental predictor can discriminate between sites versus non-sites. If we see a difference between the two sample groups, this means there is a good chance the predictive model can too. It helps to have statistical rigor behind this comparison, and the K-S Test provides this rigor in terms of a D statistic and an associated significance. For the K-S Test, we are interested in only the numeric variables.

Figure 17 is an example K-S Test outputs. It contains several pieces of important information. First, visual interpretation of the graph for 'CP\_RIVER' (path distance to nearest river) shows that the two lines (one black for sites/surveys and one blue for background) have a separation distance. This means that the site and background groupings differ with regards to their statistical distributions. This difference is measured by the D statistic listed at the top of the figure. The larger the D value (and the larger the gap between the two lines) the better. The p-value is the significance of the D statistic and should be  $p < 0.05$ . Any variable that has a p value greater than 0.05 should be excluded from modeling. In Figure 17, the 'ASP\_RNG' (aspect range) predictor variable does not discriminate background from site locations, and its high P value indicates that it should be removed from consideration.



**Figure 17: K-S Test Plots**



**Standard Scripts:**

1. Run the K-S Test on numeric variables.

```
r<-ArchModNum[,!(names(ArchModNum) %in% Categorical)]
```

This will generate a temporary dataframe called ‘r’ and call the ‘Categorical’ variable list we generated. In this case, we tell RStudio that we want to exclude all categorical variables in the ‘ArchModNum’ dataframe.

2. The next lines will display the numeric K-S Test results for all the numeric variables in the ‘r’ dataframe and plot the graphs to a .pdf file in your working directory.

```
pdf("KS_Plots")
for(col in 1:ncol(r)) {
  ks<- ks.test(subset(r[,col],r$RespVar=="0"),subset(r[,col],r$RespVar=="1"),data.name=FALSE)
  print(ks)
  plot(ecdf(subset(r[,col],r$RespVar=="0")),do.points=FALSE,verticals=T,col="blue",xlab=colnames(r)[col],
  main=paste("KS Test \n", "p-value = ", ks$p.value, "\n D = ", ks$statistic))
  lines(ecdf(subset(r[,col],r$RespVar=="1")),lty=3,do.points=FALSE,verticals=T)
}
```

3. Run the next two lines to remove the temporary dataframe and make the .pdf file available for viewing.

```
dev.off()
rm(r,col, ks)
```

**Figure 18: Numeric Results of Kolmogorov-Smirnov Test**

```
Two-sample kolmogorov-smirnov test

data: subset(r[, col], r$RespVar == "0") and subset(r[, col], r$RespVar == "1")
D = 0.37797, p-value < 2.2e-16
alternative hypothesis: two-sided

Two-sample kolmogorov-smirnov test

data: subset(r[, col], r$RespVar == "0") and subset(r[, col], r$RespVar == "1")
D = 0.20808, p-value < 2.2e-16
alternative hypothesis: two-sided

Two-sample kolmogorov-smirnov test

data: subset(r[, col], r$RespVar == "0") and subset(r[, col], r$RespVar == "1")
D = 0.070284, p-value = 0.008911
alternative hypothesis: two-sided

Two-sample kolmogorov-smirnov test

data: subset(r[, col], r$RespVar == "0") and subset(r[, col], r$RespVar == "1")
D = 1, p-value < 2.2e-16
alternative hypothesis: two-sided
```



The results in the R Console window should be similar to the example in Figure 18. You do not need to record these in your Model Report, as this procedure will also create a more useful .pdf file, called 'KS\_Plots,' that you can open in Acrobat (Figure 17). After running these lines of code, add the '.pdf' extension to the filename so that Acrobat will open it.

4. Determine which variables to remove on the basis of the Kolmogorov-Smirnov test ( $p > 0.05$ ) and use the following line to update the [RmVariableList](#). In this example, we want to exclude 'ASP\_RNG' as well as 'WETSOIL', which was removed previously. Here we generate a list by calling the names of the variables we want to remove.

```
RmVariableList<-c("WETSOIL","ASP_RNG")
```

This function generates a new list (also identified as an R 'object') that will expand as you work through the EDA procedures. This list will contain variables we want to exclude from the predictive model.



Record the variables with  $p > 0.05$  in your spreadsheet and update your record of removed variables. You will find that having a growing record of removed variables in your spreadsheet is a very handy reference.

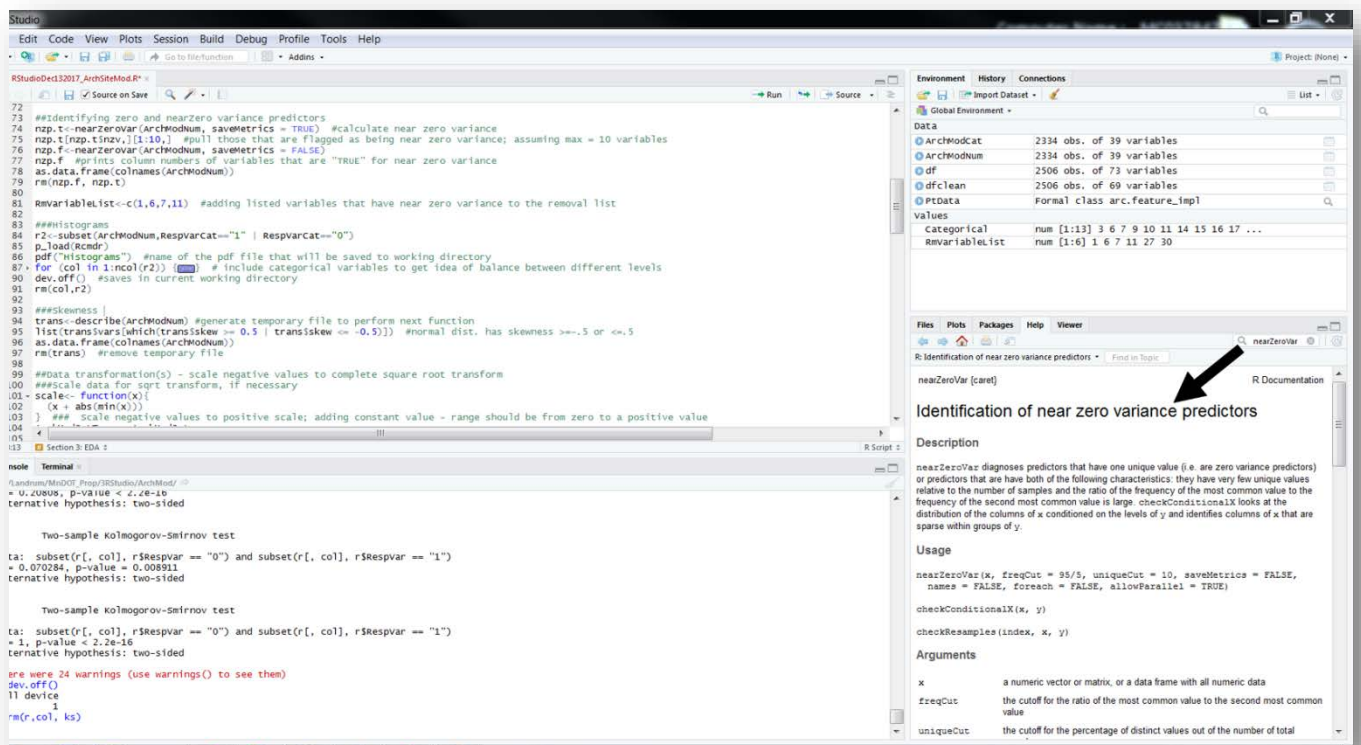
**Enhanced Scripts:** The enhanced scripts will run the K-S tests, print the results, and create the .pdf file with its extension. You do not need to rename the .pdf file. The script should also report the list of variables with  $p > 0.05$  and automatically add them to your removed variable list.

## Assessing Zero and Near Zero Variance

Predictor variables that exhibit zero or near zero variance offer very little information and can cause problems when training a predictive model. The goal is to remove these variables before modeling.

The `nearZeroVar()` function will identify predictor variables that exhibit a zero or near zero variance. To learn more about this function, reference the Help tab in the window beneath the Global Environment window. Search 'nearZeroVar' and you will see something similar to the Figure 19.

**Figure 19: Use of Help Tab**



## Standard Scripts:

1. Run the following line to generate a temporary dataframe to run the near zero variance analysis.

```
nzp.t<-nearZeroVar(ArchModNum, saveMetrics = TRUE)
```



Since the `nearZeroVar()` function requires numeric variables, we call the 'ArchModNum' dataframe. Remember, this dataframe reads categorical predictor variables as numeric. In this case, we want to **exploit the near zero variance function** to screen which categorical variables might exhibit strong imbalance between classifications. In the next subsection, histograms will provide visual confirmation of class imbalance.

2. Run the next line to generate a list of variables that exhibit zero or near zero variance.

```
nzp.t[nzp.t$nzv,][1:10,] #pull those that are flagged as being near zero variance; assuming max = 10 variables
```

Figure 20 provides an example of the near zero variance function output and indicates that two variables are flagged as having zero or near zero variance (CP\_WAT and ISLAND); up to 10 variables can be listed.

**Figure 20: Near Zero Variance**

```
> nzp.t[nzp.t$nzv,][1:10,] #pull those that are
      freqRatio percentUnique zeroVar  nzv
CP_WAT  129.7273      4.6571798  FALSE TRUE
ISLAND  308.2000      0.1293661  FALSE TRUE
NA              NA              NA     NA  NA
NA.1           NA              NA     NA  NA
NA.2           NA              NA     NA  NA
NA.3           NA              NA     NA  NA
NA.4           NA              NA     NA  NA
NA.5           NA              NA     NA  NA
NA.6           NA              NA     NA  NA
NA.7           NA              NA     NA  NA
```



Copy the `nzp.t` results to your Model Report.

3. The `rm()` function removes temporary files from the Global Environment.

```
rm(nzp.t)
```

4. Update the 'RmVariableList' list to ensure these variables are excluded from the predictive model. Be sure to edit this line prior to running so that the correct variables are removed. For example:

```
RmVariableList<-c("WETSOIL","ASP_RNG","CP_WAT","ISLAND")
```



Finally, record variables removed because of near-zero variance in your spreadsheet and add them to your Remove Variable List record.

**Enhanced Script:** The enhanced script will run the same code and report the results in the .html file. The .html file should also list the variables with near-zero variance, and the script should automatically add them to the list of removed variables.

## Histograms

Histograms serve four primary purposes for Phase 4 predictive modeling. First, histograms provide a visual interpretation of data distributions for numeric data. Few environmental predictor variables show a normal bell curve distribution; they are commonly skewed. Terrain variables, such as elevation, tend to be the exception. Second, the histograms can help validate the near-zero variance selections completed in the previous section. Third, it is important to screen the data for outliers. For example, the histograms are a wonderful tool to identify '0' values when it in fact this assignment is not a valid value but an assignment error in ArcGIS. Finally, by treating categorical variables as numeric, it is possible to use the histograms to screen for imbalances between classifications in categorical predictor variables.

**Standard Scripts:** Run lines the following lines to generate histograms, which will be output in .pdf format. After these lines have run, go into your working directory and add the '.pdf' extension to the Histograms file.

```
r2<-subset(ArchModNum,RespVarCat=="1" | RespVarCat=="0")
```

The `subset()` function generates a temporary dataframe, in this case called 'r2' to build the histograms.

```
pdf("Histograms")
```

The `pdf()` function assigns the title of the pdf ( "Histograms").

```
for (col in 1:ncol(r2)) {with(r2, Hist(r2[,col], groups=RespVarCat, scale=c("percent"), breaks=20, col="blue",  
xlab=substitute(paste(a),list(a=colnames(r2)[col])), ylab="Percent", main="Sample Histograms"))}
```

This function generates the histograms. This includes the categorical variables to get an idea of imbalances between factor levels. However, because it includes categorical variables, it throws an error message. You can ignore this.

```
dev.off()
```

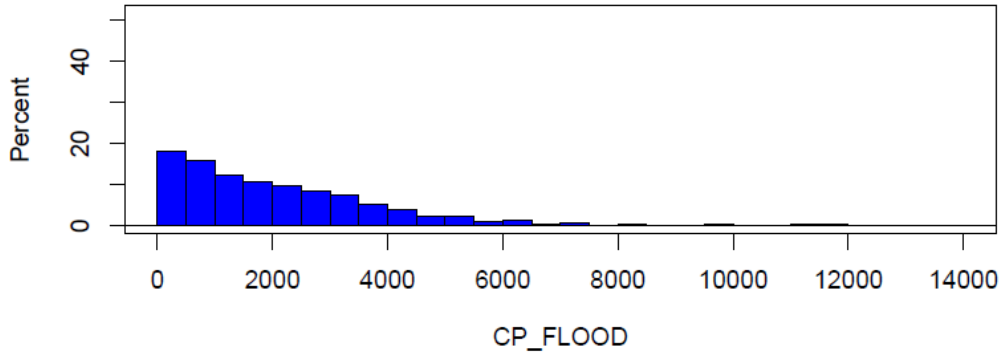
The `dev.off()` function saves the work.

Figure 21 is an example of the .pdf printout for 'CP\_FLOOD' (path distance to historic floodplain). The histograms are grouped according to site (RespVarCat=1) and background (RespVarCat = 0) for each predictor variable. It is clear from the histogram that sites in this region tend to be clustered close to floodplains.

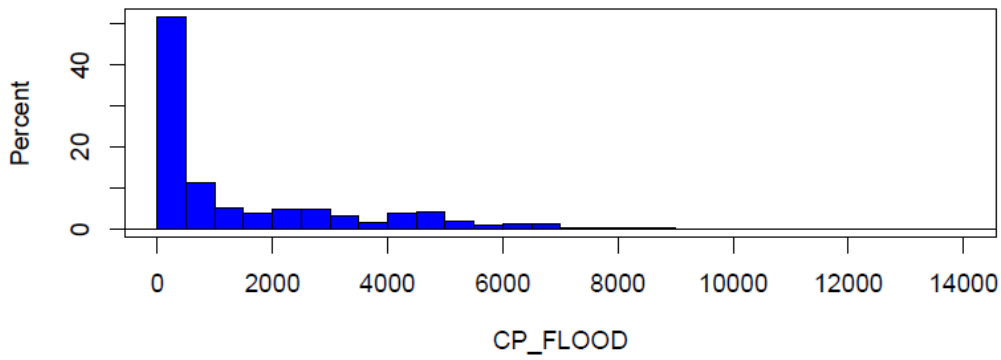
**Figure 21: Histograms for CP\_FLOOD (Path Distance to Historic Floodplain)**

Sample Histograms

**RespVarCat = 0**

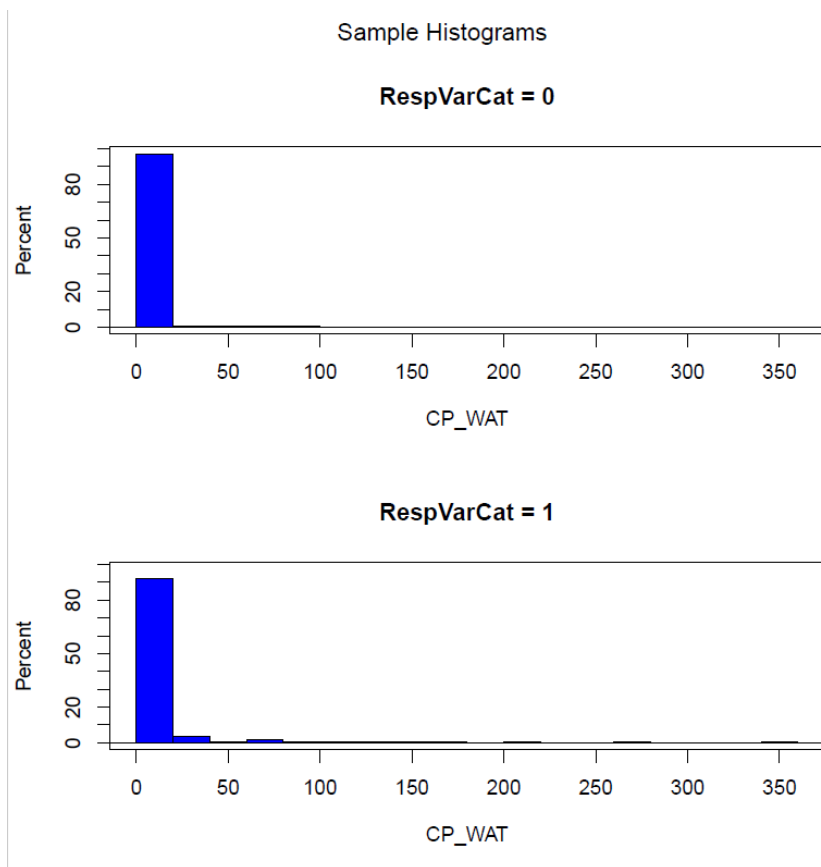


**RespVarCat = 1**



The histograms generally agree with the near zero variance assignments. Figure 22 is the histogram for 'CP\_WAT' (path distance to historic water of all types), which shows that the majority of both sites and background points are very close to some type of water

**Figure 22: Histograms for CP\_WAT (Path Distance to Historic Water of All Types)**

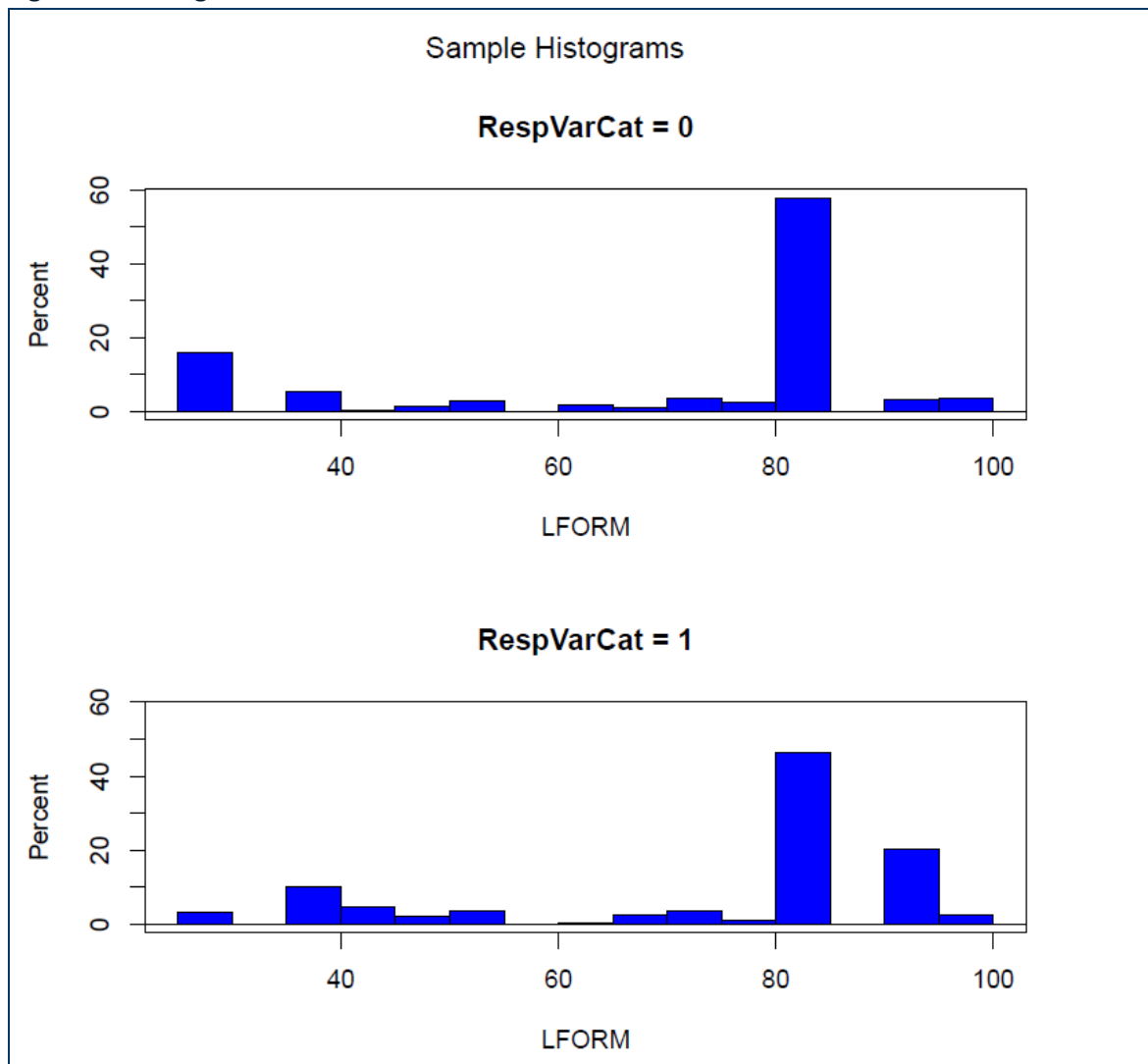


Histograms also provide a visual expression of the imbalances we have already seen in the categorical variables. In Figures 23 and 24, counts for the frequency for the landform '84' (Plain) is an order of magnitude higher than other landform classes. A predictive model may not be sensitive to less frequent classes since they have relatively very low presence. The re-classification of the very low frequency values that we did in Section 2 should help. We need to keep an eye on these variables to determine whether they help or hinder the models.

Figure 23: Frequency Count: LFORM

```
> count(ArchModNum, 'LFORM')
  LFORM freq
1      29 185
2      37  73
3      40  36
4      45  25
5      49  25
6      52  50
7      65  21
8      68  24
9      72  27
10     73  27
11     80  30
12     84 837
13     93 135
14     96  51
```

Figure 24: Histogram of LFORM Values





**Enhanced Script:** The enhanced script creates the histograms in the output .html file, along with all of the other model results. There will be no separate histogram .pdf file.

### Skewness (optional)

The skewness functions are optional and will flag predictor variables that exhibit a skewed data distribution. This step is optional because ‘tree’ methods (e.g. bagging and random forest) can perform generally well with skewed environmental data, therefore, it is not advisable to transform numeric data to perform Phase 4 predictive modeling. Applying a universal or ‘cookie cutter’ transformation, such as the square root, sine and/or cosine transforms, to skewed data can introduce error in the predictive model if the transform is ineffective. If, however, the User wishes to utilize other modeling procedures, such as logistic regression, square root, sine and cosine transformations may be needed for skewed variables. Additional scripts were developed for this eventuality. They can be made available to anyone interested, but they have not been tested. If used, the square root transform should be applied to non-directional numeric predictor variables and the sine and cosine transforms should be applied to directional numeric predictor variables (Oehlert et al. 2007a; Oehlert et al. 2007b).

#### Standard Scripts:

1. Run `trans<-describe()` to create the temporary file.

```
trans<-describe(ArchModNum)
```

2. Run `list()` to generate a list of skewed predictor variables, identifying them by their column number in the dataframe. A normal distribution has skewness  $\geq -0.5$  AND  $\leq 0.5$ .

```
list(trans$vars[which(trans$skew >= 0.5 | trans$skew <= -0.5)])
```

Your output will look something like this – a list of the column numbers of skewed variables:

```
[1] 1 2 3 7 8 9 10 12 17 19 20
```

3. The next two functions will 1) display column numbers along with their headers to easily identify the skewed variables by name and 2) remove the temporary file.

```
as.data.frame(colnames(ArchModNum))  
rm(trans)
```

Use this list to determine, from the column numbers, which variables are skewed.



Copy your results to your Model Report.

**Enhanced Script:** The enhanced script runs the same code and provides the same output.

## Collinearity

The predictive models recommended for Phase 4 make several assumptions. One important assumption is that the predictor variables are independent, i.e. not correlated. If the predictor variables are independent, they offer unique information to train and optimize the predictive model, thereby generating a more informed and ‘well rounded’ predictive model. Additionally, information redundancy is a sign that the research team is investing time, labor and resources to generate multiple versions of the same information in their environmental data or predictor variables. There are practical and cost-driven incentives to reduce information redundancies.

Oftentimes collinearity, or information redundancy, emerges when ArcGIS predictor variables are created using the same measure but at different geographic scales. For example, the ‘TPI250’ (Topographic Position Index within a radius of 250 meters) is strongly associated with ‘TPI500’ (Topographic Position Index within a radius of 500 meters) but not with ‘TPI5MI’ (Topographic Position Index within 5 miles). Until trying these variables at different scales, there is no way to determine which scales are correlated and which add new information to the model. Collinearity may also occur when one ArcGIS predictor variable is the source for generating similar or secondary predictor variables. Of course all terrain variables are derived from ‘elevation’ (ELEV), though they are not all correlated with one another. An evaluation of the correlations between the various terrain variables in different types of landscapes would be informative.

The objective of this subsection is to test and identify information redundancies, or collinearity, between predictor variables. This section of the modeling script generates several tables and figures helpful for understanding the correlations between numeric predictor variables. This statistical method is not applicable to categorical variables. Correlations between categorical variables are discussed under the Chi-squared Test of Independence below.

**Standard Scripts:** The first step in this analysis is to perform Spearman’s Rank Correlation test to identify correlated variables. This section of the script produces a correlation matrix and graphical output in .pdf format. Correlation coefficients with a p value < 0.05 indicate that there is a significant relationship. Positive coefficients indicate a direct relationship between two variables; a negative sign indicates an inverse relationship between two variables. In general, if a correlation coefficient exhibits a value greater than 0.70 or less than -0.70 the two variables offer redundant information. This is a rule of thumb, however, and you can change this cutoff value to be less conservative (e.g. change to 0.9). Note that the Spearman correlations will tell you which variables are correlated with one another, but not which variable of a correlated pair to remove. The VIF test (below) will help with that decision.

After calculating the Spearman’s Rank Correlation, the script calculates the Variance Inflation Factor (VIF). This analysis supports decisions regarding which variables are best to exclude from the model. The VIF provides a simple measure of collinearity using a multivariate regression approach. In layman’s terms, each predictor variable takes its turn serving as the response variable, then is fitted to all the remaining predictor variables using a generalized linear regression model. The VIF is calculated using the r-squared value of the fitted generalized linear model for that predictor variable against all other predictor variables. The smaller the VIF the better. A common rule of thumb is if the VIF is greater than 5-10, the variable is introducing collinearity to the regression model. The VIF is applicable for numeric variables only.

## Spearman's Rank Correlation Coefficients

1. First, we need to exclude categorical variables and variables we have added to the 'RmVariableList' up to this point. The `append()` function generates a temporary list that appends the variables in the 'Categorical' list with those variables listed in the 'RmVariableList', so, basically, we are compiling these lists into one.
2. Next, run the following line to generate a temporary dataframe to calculate the Spearman's rank correlation coefficients (SRCC) on numeric predictor variables.

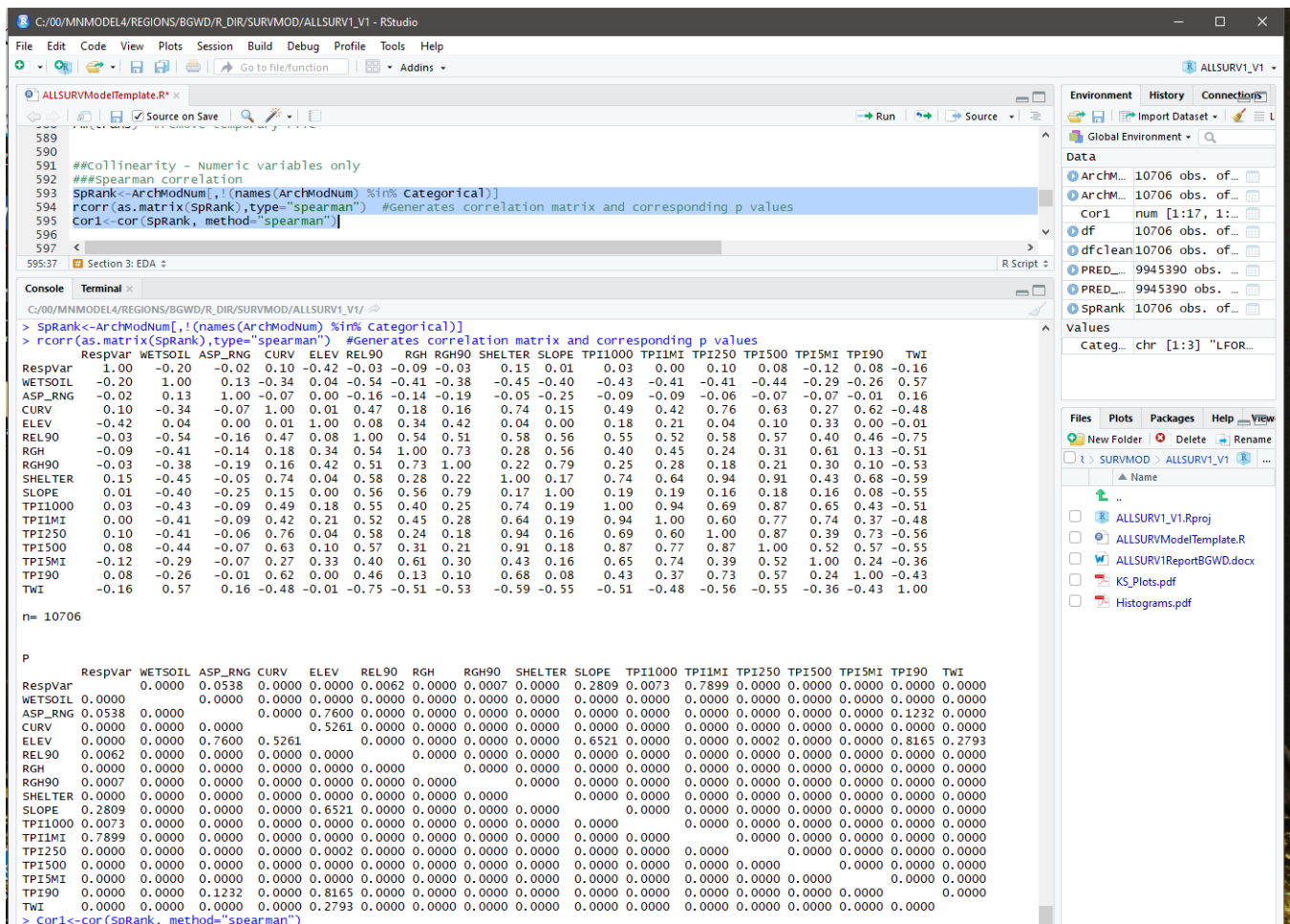
```
SpRank<-ArchModNum[!(names(ArchModNum) %in% appendlist)]
```

3. The `rcorr()` function prints the correlation matrix in the Console, as shown in Figure 25 below, along with the corresponding significance (p values).

```
rcorr(as.matrix(SpRank),type="spearman")
```

```
Cor1<-cor(SpRank, method="spearman")
```

Figure 25: Spearman Correlation Coefficient Matrix





Copy this matrix to your Model Report.

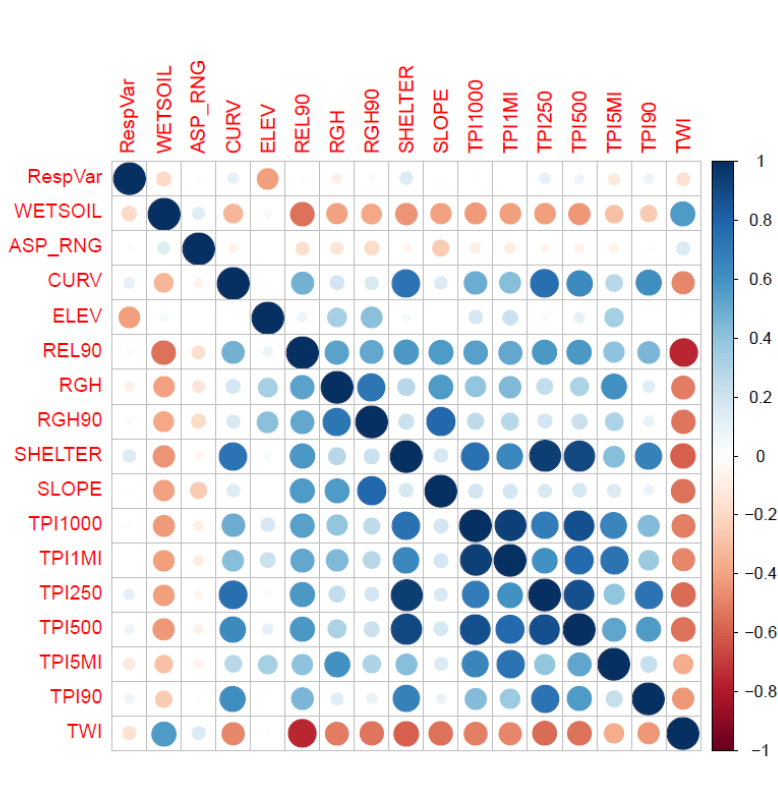
- Sometimes visual tools can convey the same information in a simpler form. After generating the correlation matrix, you can run the following lines to save two pdf figures to your working directory.

```
pdf("Figure 1 Site Predictor Variable (Numeric) Spearman Rank Correlation Plot")
corrplot(Cor1,method="circle")
dev.off()
```

```
pdf("Figure 2 Site Predictor Variable (Numeric) Spearman Rank Correlation Coefficients")
corrplot(Cor1, method="number", number.cex=.6)
dev.off()
```

In both figures, direct correlation coefficients are represented by cooler colors and inverse correlation coefficients are represented by warmer colors. The first pdf is shown in Figure 26. The size of the dot in this figure represents the magnitude or strength of the correlation. The diagonal represents the correlation coefficient of a predictor variable with itself (perfect correlation). The second pdf replaces the dot with the correlation coefficient. Be aware that these diagrams become less legible as more variables are added to the model.

Figure 26: Spearman Rank Correlation Plot



The 'RespVar' is irrelevant in this figure because it is the numeric version of the categorical binary response variable. However, it may be interesting if any single variable is strongly correlated with the response variable.

5. Run `findCorrelation()` to print the column number of variables with a correlation coefficient exhibits a value greater than .070 or less than -0.70. Note that if you do not agree with this rule of thumb you can change the cutoff value from +/- 0.7 to a higher absolute threshold.

```
findCorrelation(cor(SpRank, method = "spearman"),cutoff=0.7, exact=FALSE)
```



If you are interested in the relationships between variables, you may wish to record pairs of variables with a Spearman correlation coefficient > 0.7 in your worksheet. Otherwise, it will be helpful to print out the correlation matrix and highlight the pairs of correlated variables. You will be able to use this information in the VIF analysis later.

6. Finally, run the next two lines to describe the data and remove the temporary files.

```
psych::describe(SpRank)
rm(SpRank,Cor1)
```

The `rm()` function cleans up the Global Environment window, while the `psych::describe()` function refreshes the memory of the basic stats.

### Variance Inflation Factor (VIF)

The following functions calculate the VIF and display the results in the Console window.

1. The following line will build a temporary generic regression model called 'vif.mod':

```
> vif.mod<-glm(ArchModCat$RespVar~.,data=ArchModCat,! (names(ArchModCat) %in% appendlist)),fa
mily=binomial(logit))
```

In this function line, we call the ArchModCat dataframe using '`data=ArchModCat [!(names(ArchModCat) %in% appendlist)]`' and we are telling R to exclude the variables in the '`appendlist`'. Another way to say this, we include all variables in this dataframe except those that are listed in the '`Categorical`' and '`RmVariable`' lists.

The '`family=binomial(logit)`' argument tells RStudio the type of generalized linear model that we are running. The '`ArchModCat$RespVar~.`' identifies the response variable and the '`~.`' indicates that all variables should be used that are identified by '`data=`' argument.

2. The `formula()` function will display the extended version of the generalized linear model ('glm') regression formula.

```
formula(vif.mod)
```

Record this and make sure the regression model is not including unwanted variables (e.g. categorical) or excluding necessary variables (e.g. numeric variables). It is also important to make sure that the response variable is not included as a predictor variable.

3. The `summary()` function will print the results of the fitted generalized linear model. The results of the generalized linear regression model provide a first indication as to which numeric variables are likely important for predicting site versus background locations. You can screen variable importance by looking at their significance.

`summary(vif.mod)`



If you are interested in performance of the individual variables, you may wish to keep a record of this summary. The significant codes are provided at the bottom of the display and are recognized by a '\*'. This significance can change as variables are added or removed from the multivariate regression model. Any variable with at least one asterisk is considered significant at the 95 percent confidence level.

4. The `vif()` function calculates the VIF using the fitted `vif.glm` model. Variables with a VIF greater than 10 are problematic.

`vif(vif.mod)`

**Figure 27: VIF Values**

```
> vif(vif.mod)
      WETSOIL  ASP_RNG    CURV    ELEV    REL90    RGH    RGH90  SHELTER    SLOPE  TPI1000  TPI1MI  2
1. 478650  1. 049786  2. 557744 20. 958695  4. 350034  3. 512037 121. 084718  35. 319704  96. 940366  25. 448641 16. 059462
```



Copy the VIF scores to your Model Report.

In Figure 27, several variables show a VIF above 10. These are 'ELEV', 'RGH90', 'SHELTER', 'SLOPE', 'TPI1000', 'TPI1MI', 'TPI250', and 'TPI500'. These may include variables not flagged by Spearman. Removing one or more of these variables will reduce the VIF for the other variables. Removing variables is a stepwise, iterative process where you drop one or more variables from the regression model and recalculate the VIFs. Deciding which variables to exclude and in what sequence will depend on professional judgement, the VIF value, results from the Spearman correlation analysis, and general trial and error.

5. The next step is to remove variables causing collinearity and run the model again. Note that in the lines of code below, everything in red is an edit that will be unique to each run of the 'vif' function.

```
vif.mod2<-glm(ArchModCat$RespVar~.-RGH90-TPI1MI-
TPI250,data=ArchModCat[,!(names(ArchModCat) %in% appendlist)],family=binomial(logit))
```

```
formula(vif.mod2)
summary(vif.mod2)
vif(vif.mod2)
```

The first line indicates which variables were removed from the regression model for the sake of establishing a sound modeling procedure. The variables are listed with a subtraction sign preceding their name in the regression argument.

You will note in this example that not all variables with values greater than 10 in the original VIF analysis were removed. You will necessarily try several VIFs with different combinations of variables removed to determine which perform best. These iterations demonstrated that removing 'RGH90', 'TPI1MI', 'TPI250', and 'SHELTER' provided the best results.



Record your results in your Model Report. Figure 28 shows the results of a run producing no VIF values greater than 10.

**Figure 28: Results of Third VIF Run**

```
> vif(vif.mod3)
WETSOIL ASP_RNG CURV ELEV REL90 RGH SLOPE TPI1000 TPI500 TPI5MI TPI90 TWI
1.476807 1.035947 2.026769 1.296262 3.207840 2.976401 2.253777 5.664190 6.101230 2.616887 1.992405 1.579037
```

- Repeat step 5, experimenting with removing different combinations of the originally flagged variables, until your VIF results have no values greater than 10. Be sure to sequentially number your VIF models and record all of your results in your Model Report.

Use your Spearman correlation results, as recorded in your Model Report, to help you determine which combinations of variables to test in VIF. For example, Spearman showed that 'RGH90' was correlated with both 'RGH' and 'SLOPE', while 'RGH' was correlated with only 'RGH90'. This might indicate that removing 'RGH90' would solve more problems than removing 'RGH'. In any case, there is no reason to remove both at the same time.



When you have determined the best set of variables to remove, record them in your worksheet.

- Update the `RmVariableList()` function in your script and your modeling record.

```
RmVariableList("WETSOIL","ASP_RNG","VIFVAR1","VIFVAR2")
```



Record the current complete version of your Remove Variable List in your Model Report.

Note that the variables listed include those identified for removal by:

- The search for NULL values, including WETSOIL if it has any NULL values.
- The K-S Test
- The Near Zero Variance test



- And the Spearman/VIF tests for correlation

7. Run the `rm()` function to remove the temporary files. Be sure to list all of the VIF iterations you ran.

```
rm(vif.mod,vif.mod2,vif.mod2)
```

**Enhanced Script:** The enhanced version of the script effectively automates the Spearman's and VIF procedures for the user. The .html file does not print the entire correlation matrix, but instead lists only pairs of variables correlated with coefficients greater than 0.70 or less than -0.70. The script will then run the VIF analysis and report all VIF scores > 10. The automatic second iteration repeats the correlation and VIF steps, removing single variable with the highest VIF score from the first analysis. The script will continue iterations of Spearman and VIF, removing one variable at a time, until reaching the point where no VIF scores exceed 10. Variables removed via these iterations will be added to the list of variables to remove. The updated RmVariableList will be printed at the end of this set of iterations.

### Chi-squared Test of Independence

The objective of this step is to determine which categorical variables in our dataset are important and informative. Categorical variables are present in vegetation and geomorphic data. Many of the categorical variables result from a classification system that uses professional judgement and visual interpretation of available geospatial data layers, such as aerial imagery and digital terrain data. Collinearity will result from the nature in which categorical variables are classified. In some cases, collinearity can be confounding, meaning that if a soil type is classified using digital elevation data, then soil texture is classified based on soil type, all three variables are associated directly and indirectly.

Our three categorical variables are undoubtedly related. The geomorphic variables, 'LFORM' and 'LSCAPE', are part of a hierarchical geomorphic classification system where landforms ('LFORM') are portions of landscapes ('LSCAPE'). However, many landforms can be found in more than one landscape. For example, the ubiquitous landform 'Plain' may be found in Stagnant Ice landscapes, Active Ice landscapes, Lake Plain landscapes, etc. For this reason, the 'LSCAPE' variable may provide different information than the 'LFORM' variable. We expect historic vegetation type ('VEGMOD') to be related to 'LFORM' and 'LSCAPE' because the two geomorphic variables were predictors in the model used to create 'VEGMOD'.

The Chi-square Test of Independence will help us understand in what ways these variables are related. It assesses relationships between two categorical variables using a contingency table. We repeat this three times to evaluate relationships between the three categorical predictor variables and each other and another three times to evaluate the relationships between the response variable (site/survey presence or absence) and the categorical predictor variables.

All of the categorical variables tested while developing this statistical procedure showed significant correlations. In such cases, it is best to rely on the predictive model to help assess which categorical variables should be removed based on variable importance and model performance.



## Standard Scripts:

1. First use the `table()` function to generate a temporary table for comparing two categorical variables (i.e. 'LFORM' and 'LSCAPE').

```
tbl<-table(ArchModCat$LFORM,ArchModCat$LSCAPE)
```

2. The `chisq.test()` function uses the temporary table generated by `table()` and will display the results of the test in the Console window.

### Pearson's Chi-squared test

```
data: tbl1
X-squared = 6530.2, df = 143, p-value < 2.2e-16
```

If the p value is less than 0.05 the two variables are correlated at the 95 percent confidence level.



Copy these results to your Model Report.

3. The `tbl` function displays the contingency table, allowing visual assessment of how classes are related between the two variables. For example, class 84 in the 'LFORM' variable is redundant with classes 14 and 18 in the 'LSCAPE' variable. In other words, the landform 'Plain' is more often found on 'Collapsed Sand Plain' and 'Glaciofluvial' landscapes in this region than on other landscapes.

```
> tbl1
```

	10	11	12	13	14	16	17	18	22	24	26	27
29	0	0	0	6	110	3	0	48	0	18	0	0
37	0	0	0	0	0	73	0	0	0	0	0	0
40	0	17	0	0	0	0	0	18	0	0	0	1
45	0	0	0	0	0	0	25	0	0	0	0	0
49	0	4	0	0	0	0	21	0	0	0	0	0
52	10	0	0	0	1	0	0	1	11	9	18	0
65	0	0	0	3	15	0	0	3	0	0	0	0
68	0	0	1	2	2	0	0	3	0	16	0	0
72	0	27	0	0	0	0	0	0	0	0	0	0
73	0	0	0	0	0	0	27	0	0	0	0	0
80	0	5	0	0	0	0	3	21	0	0	0	1
84	25	12	0	72	379	0	0	297	18	34	0	0
93	0	9	0	0	0	0	4	96	0	0	0	26
96	0	0	50	0	0	0	0	1	0	0	0	0



Copy the table to your Model Report.

4. `rm(tbl)` removes the temporary table.
5. Repeat for other variable pairs.

**Enhanced Script:** The enhanced script runs the same code as the standard scripts and produces the same output.

## Section 4: Predictive Model Training and Testing

---

The general workflow for Step 4 is as follows:

1. Generate a training and testing dataset
2. Fit a random forest model to the training dataset
3. Optimize fitted random forest
4. Assess model performance by applying the fitted model to the test dataset
5. Repeat Steps 1-4 four times, creating, optimizing, and evaluating four different models. This is known as a 'jackknife' procedure.
6. Fit a final random forest model to the entire sample dataset.

R scripts for BIC stepwise logistic regression, naïve Bayes, multinomial logistic regression, trees, and bagging are available in the `\MNMODEL4\TOOLS\R\Modeling Scripts\PredictiveModel\OptionalAdditionalProcedures` folder. Their use is not recommended for Phase 4 predictive modeling, but they may be useful for projects exploring relative performance of different modeling procedures.

### Step 1: Generate Training and Testing Datasets

It is possible, and even desirable, to develop a model using the entire training dataset of sites or surveys and background points. However, it would be impossible to evaluate the resulting model using statistical tools. For that reason, we randomly divide the total training sample into two datasets: one group of records trains the predictive model and the second is used to test the model. We do this four times so that we can evaluate four models developed from different subsets of the data.

**Standard Scripts:** The standard scripts in the `\MNMODEL4\TOOLS\R\Modeling Scripts\PredictiveModel\TEMPLATE\INITIAL` folder do not perform the jackknife procedure. They each produce and test one version of the model based on a 75/25 split of the data. These were used to develop some of the initial site models and the survey models (Hobbs 2019b). The `ALLSITEModelTemplateJack.R` script in the `\MNMODEL4\TOOLS\R\Modeling Scripts\PredictiveModel\TEMPLATE\JACKKNIFE` folder is the correct template for the jackknife procedure for the site model without soils data. It would need to be edited to apply to survey data or to include soil variables. This jackknife script was used to develop and test the procedure, but all jackknife modeling for MnModel Phase 4 was done using the enhanced script.

#### *Split the Data*

1. The next few lines randomly split the training data into two groups. Because there is a random component to this split, we first must set a 'seed' such that if we want to reproduce this split in the

future we can. We set a seed using the `set.seed()` function. It does not matter what number you choose for the seed, but you must use the same number to reproduce the split in the future. To produce a different split, you must use a different number.

```
set.seed(2)
```

Run the `createDataPartition()` function to generate the data split.

```
RMmodel<-createDataPartition(ArchModCat$RespVar,p=.75,list=FALSE)
```

The `createDataPartition()` function uses a random generator to split the data into training and testing datasets. In the `createDataPartition()` function the '`p=.75`' argument means we are doing a 75/25 split where a randomly selected 75 percent of the samples will be allocated for training the predictive model and the remaining 25percent will be used to test the model. The test dataset will provide an unbiased assessment of model performance because the model never observes these test data during model fitting.

```
RMTrain<-ArchModCat[RMmodel,! (names(ArchModCat) %in% RmVariableList)]
```

```
RMTest<-ArchModCat[-RMmodel,! (names(ArchModCat) %in% RmVariableList)]
```

These two lines create the training and testing dataframes, called '`RMTrain`' and '`RMTest`'. Note we are excluding the variables we added to the '`RmVariableList`' list during EDA.

```
psych::describe(RMTrain)
```

The `psych::describe(RMTrain)` function is a QAQC step to ensure the variables in the '`RmVariableList`' were removed and that the training data records for the training dataset equate to 75 percent of the total sample count.



Copy these results to your Model Report.

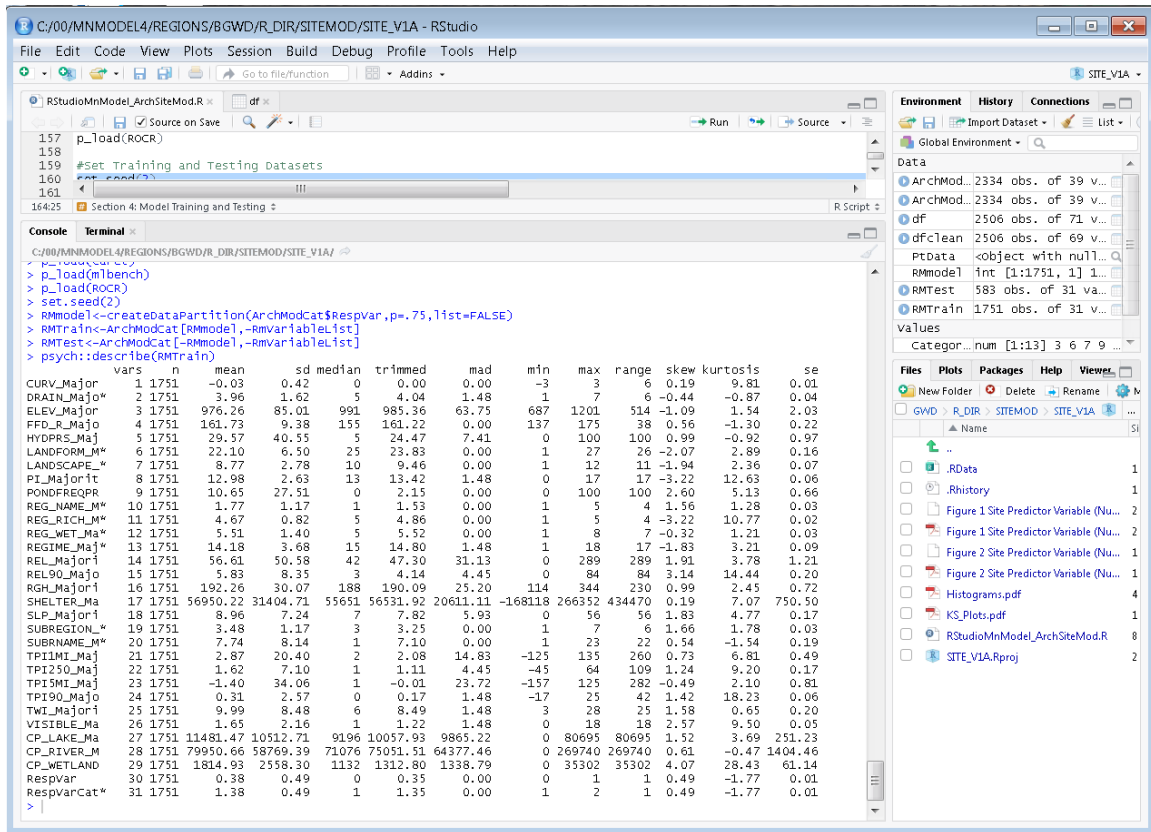


There is nothing magic about a 75/25 split. We could do a 90/10 split, particularly if we want to increase the number of sites in the training data for each model. We could also do a 50/50 split, as we did in MnModel Phase 3, or use any other ratio.

### *Verify Correct Split*

In the example below, the number of records in the training dataset is 1751 (Figure 29). When we described ArchModCat prior to this step, the number of records was 2334. Since 1751 is 75 percent of 2334, we can verify that the sample was divided into the correct proportions.

Figure 29: Generate and QC Training Dataset



### Compare Factor Levels in Training, Testing, and Predictive Datasets

When you split the data, it is possible that factor imbalances will worsen and that they could prevent the model from running. It is important to perform this quality control step now to identify and, if necessary, correct potential problems.

Repeat the `count()` function procedures for all three categorical predictive variables ('LFORM', 'LSCAPE', 'VEGMOD') for the training, testing, and predictive data. Also run `count()` for the categorical response variable ('RespVarCat') for the training and testing data.



Copy the results of each into the appropriate table in your model report.

Problems to look for:

- If the frequency of any category in the training data falls below 10. R does not seem to care if there are fewer than 10 occurrences of the predictive factors in the test data, or even if the factor level is missing from the test data, though this is not ideal for evaluation of the model.
- If you have any category in the testing data that you do not have in the training data. It is not desirable to have values in the training data that are not in the testing data, but that will not prevent the subsequent procedures from running.
- If the number of sites or surveys in the training data becomes very small.

- If you have a different number of factor levels in the training and predictive sets. The final model will not run if the training and predictive data have different factor levels or different factors. This is of concern only if you plan to run one of these 75 percent models on the predictive data. If you run only the final model, based on the entire sample dataset, on the predictive data, you should already have rectified any differences in factor levels between the two datasets.

## Resolution



If you encounter any of these problems, you **cannot proceed**.

Imbalances in the predictive variables indicate you need to re-assign values (assign the factor with low frequency to a different value with higher frequency). You have to **start a new model** to do this, as re-assigning values after factors are defined creates empty factor levels. Remember that if you re-assign any values in the training data, you must also re-assign these values in the prediction data.

Imbalances in the response variable are more difficult to resolve. One possible solution, if the total number of sites/surveys in the region should be sufficient for modeling, would be to start a new model and set a different seed in hopes to achieve a better split of the data. If there appear to be too few sites/surveys for modeling, you may need to combine this region with an adjacent region.

**Enhanced Script:** The enhanced script splits the data and runs the factor counts using the same procedures, reporting results in the .html file.

## Step 2: Train a Random Forest Model

The R `randomForest()` function provides the following built-in capacities:

- Variable importance assessment
- Bootstrap measure of model error, called an out of bag error (OOB)
- Parameter optimization
- Classification or regression modeling

The Random Forest model is known as an ‘off the shelf’ model because it is robust against most data and is user-friendly for non-statisticians.

The User should be aware of two parameters when training a Random Forest model.

- The `mtry` parameter defines the number of predictor variables the model will randomly sample at each tree node split. This parameter establishes a main difference between Random Forest and bagging. For regression modeling, as applied here, the ‘`mtry`’ default is  $p/3$ , where  $p$  = the number of model variables. Using this default,  $1/3$  of the variables will be tried at each node.
- `ntree` is a model parameter that defines the number of trees to fit the model. The default is 500 trees.

It is possible to optimize both parameters to improve overall model fit.

## Standard Scripts:

1. Fit model to training data:

```
set.seed(22)
```

Because there is a random component to the Random Forest model, it is important to first use the `set.seed()` function to ensure the model is reproducible. Again, the number you use for the seed is not important. What is important is that you use the same number to reproduce the same model in the future.

The `randomForest()` function fits a generic random forest model to the `RMTrain` dataset.

```
tree.rf<-randomForest(RMTrain$RespVar~.-RespVarCat -RespVar,data=RMTrain, importance = TRUE, proximity=TRUE)
```

This syntax produces a generic Random Forest model called `'tree.rf'` using the `'RMTrain'` dataset. Notice we are subtracting the response variable from the model (`'RespVarCat'` and `'RespVar'`). Because this is a generic model, the User opts for default parameter settings (`mtry= p/3` and `ntree= 500`). Figure 30 illustrates the expected results.

**Figure 30: Tree.rf Results**

```
> tree.rf
Call:
randomForest(formula = RMTrain$RespVar ~ . - RespVarCat - RespVar, data = RMTrain, importance = TRUE, proximity = TRUE)
  Type of random forest: regression
    Number of trees: 500
No. of variables tried at each split: 4

  Mean of squared residuals: 0.08895588
    % Var explained: 54.25
> |
```



Copy the `tree.rf` results to your Model Report for comparison to subsequent trees.

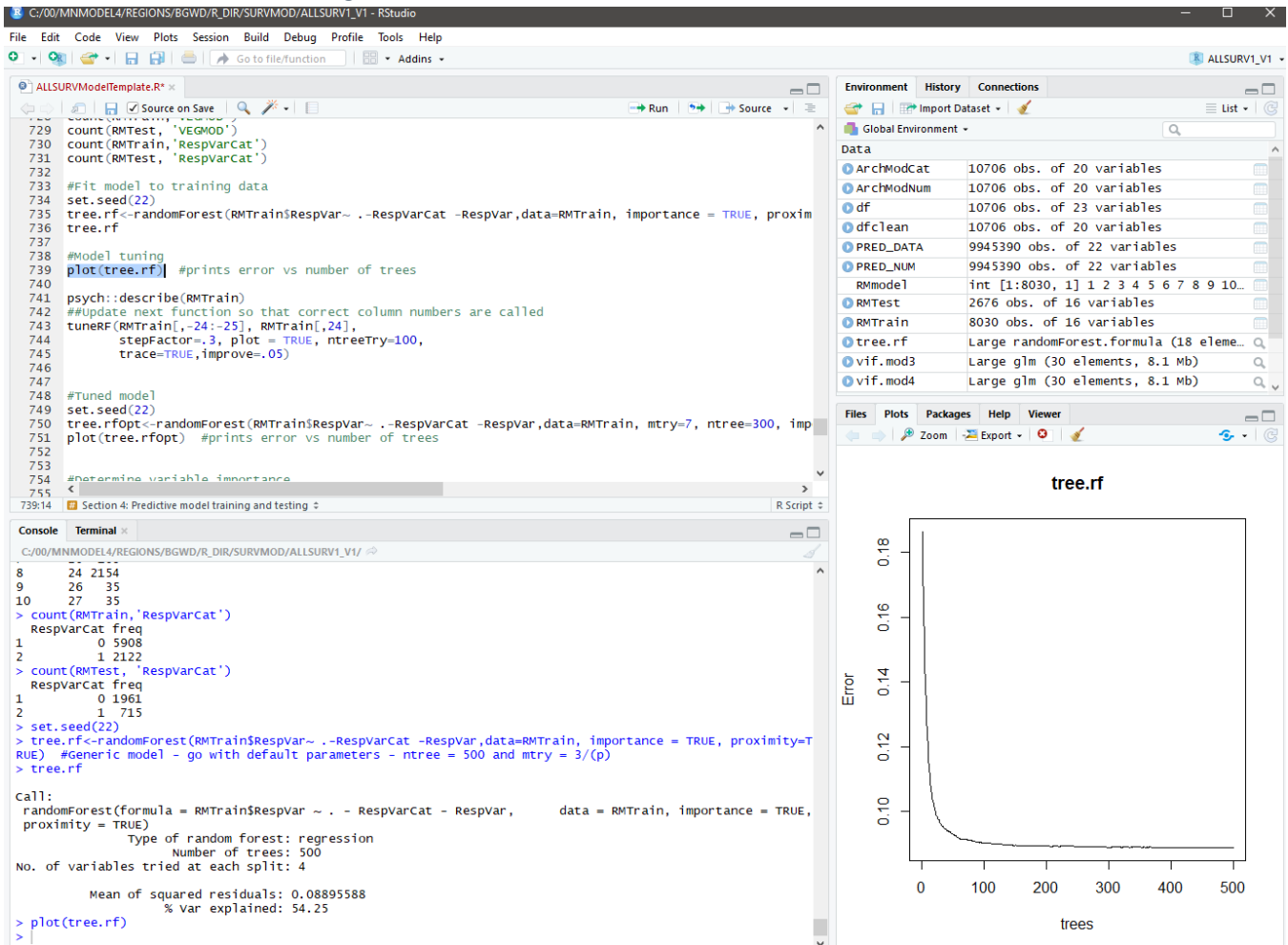
**Enhanced Script:** The enhanced script will run the `tree.rf` fitting functions and, in the same chunk of script, begin the next section for model fitting by running `plot()`

### Step 3. Optimize Model Fitting Parameters

You may optimize your random forest model by modifying the default `ntree` and `mtry` values. These functions are optional but can prove useful. These lines provide outputs for model tuning.

The `plot()` function generates a plot to help assess an optimal number for the `ntree` parameter. In Figure 31, the out of bag model error for the `'tree.rf'` model is plotted against the number of trees. The plot will appear in the 'Plots' window in RStudio. The error significantly drops until reaching approximately 50 trees, then declines infinitesimally. It is advisable to have more than 50 trees for building a model. In this case, a minimum of 300 seems sufficient.

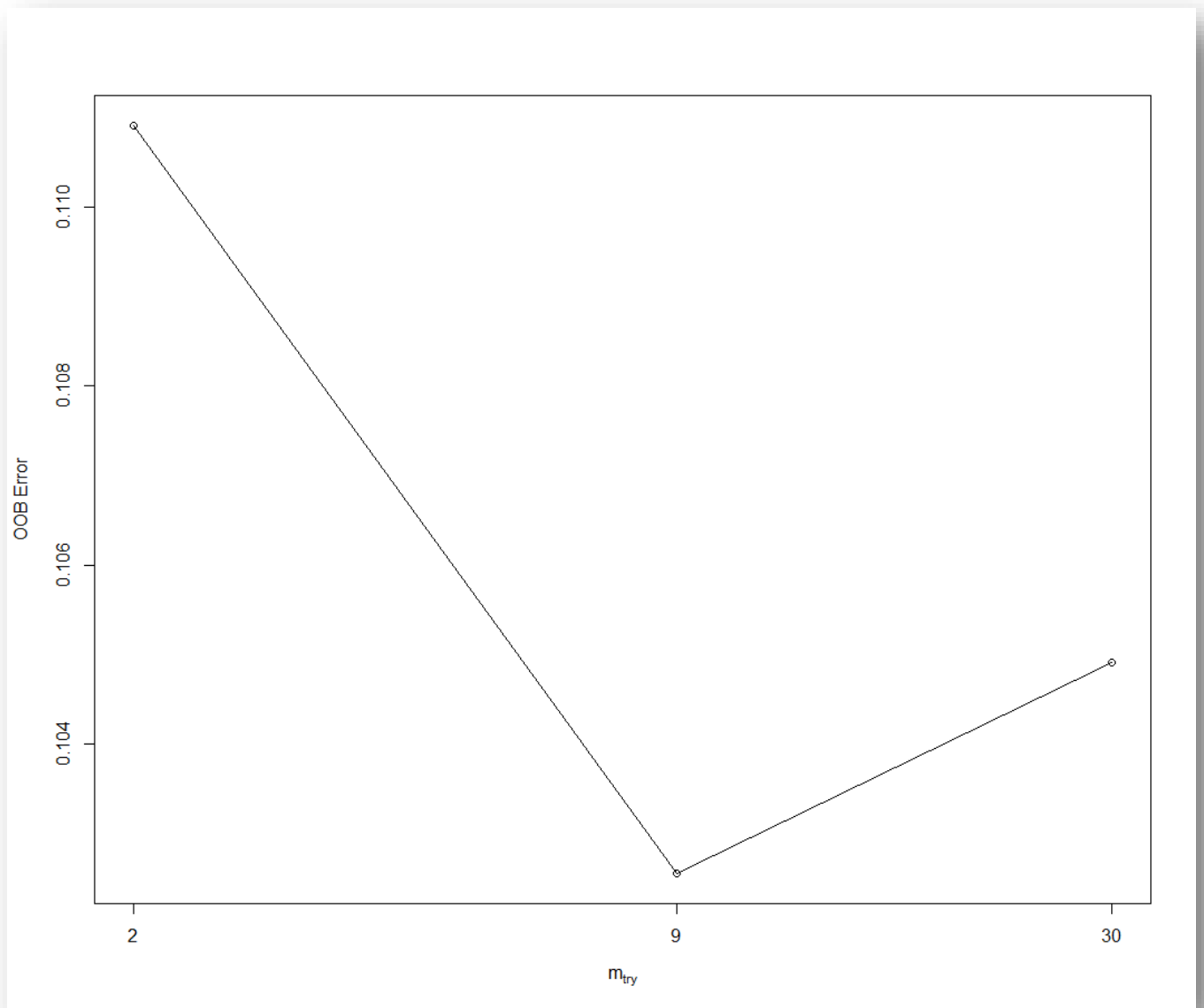
Figure 31: Tree.rf Plot for Determining ntree



The `tuneRF()` function determines an optimal `mtry` number. The required syntax and variable substitutions are explained in the next section.

Figure 32 is produced using the `tuneRF()` function. It shows the model error (y-axis) versus the `mtry` values (x-axis). The `mtry` value with the lowest model error is 9 and is optimal for the 'RMTrain' model.

Figure 32: Results of tuneRF() function



Several `mtry` values and their associated out-of-bag (OOB) errors are displayed in the RStudio console (Figure 33).



**Figure 33: mtry Values and OOB Errors from tuneRF**

```
> tuneRF(RMTrain[,-30:-31], RMTrain[,30],
+       stepFactor=.3, plot = TRUE, ntreeTry=100,
+       trace=TRUE,improve=.05) #Update this so that the right column represents response variable
mtry = 9  OOB error = 0.101775
Searching left ...
mtry = 30  OOB error = 0.1056295
-0.03787272 0.05
Searching right ...
mtry = 2  OOB error = 0.1095197
-0.07609699 0.05
  mtry  OOBError
2      2 0.1095197
9      9 0.1017750
30     30 0.1056295
```

The out of bag error (OOB error) is calculated using each of the 100 ‘*ntrees*’ used to build the Random Forest model. Each tree results from a bootstrap sample of the training dataset. The samples not included in the bootstrap sample are used to calculate the prediction error for that tree. The OOB error for regression is equivalent to the root mean squared error (RMSE) and provides an overall assessment of model training error. The smaller the OOB error, the better. However, the *mtry* value should not be larger than the number of variables in your model at this point (remembering that you have removed some variables during EDA). If you use a larger *mtry* value, the model will create dummy variables by re-using variables when splitting nodes.

The next step is to fit a second random forest model, called ‘*tree.rfOpt*’ tuned to the optimal *ntree* and *mtry* parameters. Note that when running this model, you use the same ‘*seed*’ value as your previous model but edit the *mtry* and *ntree* values based on the results of the previous steps.

### Standard Scripts:

1. Generate a plot to help assess an optimal number for the *ntree* parameter:

```
plot(tree.rf)
```



Copy the resulting plot to your Model Report.



Record the minimum *ntree* value in your worksheet.

2. Run `psych::describe()` to display the predictor variable names and their column numbers.

```
psych::describe(RMTrain)
```

Reference the column numbers for the *RespVar* and *RespVarCat* response variables (Figure 34). In the example in the next step, **1** is the column number for *RespVar* and **16** is the column number for *RespVarCat*.

Figure 34: Describe() Results for Determining Response Variable Column Numbers

```

> psych::describe(RMTrain)

```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
RespVar	1	8030	0.27	0.44	0.00	0.21	0.00	0.00	1	1.00	1.05	-0.91	0.00
LFORM*	2	8030	16.58	5.05	19.00	17.80	0.00	1.00	21	20.00	-1.82	1.80	0.06
LSCAPE*	3	8030	7.38	1.73	8.00	7.83	0.00	1.00	10	9.00	-2.42	5.06	0.02
WETSOIL	4	8030	0.39	0.49	0.00	0.37	0.00	0.00	1	1.00	0.44	-1.81	0.01
ASP_RNG	5	8030	4.85	2.55	5.00	4.83	2.97	1.00	9	8.00	0.04	-1.23	0.03
CURV	6	8030	0.26	3.66	0.00	0.11	1.48	-49.00	35	84.00	-0.16	23.58	0.04
ELEV	7	8030	988.58	78.40	998.00	994.80	68.20	687.00	1215	528.00	-0.99	1.77	0.87
REL90	8	8030	4.77	7.04	2.00	3.33	2.97	0.00	94	94.00	3.40	20.35	0.08
RGH	9	8030	189.58	30.37	184.41	186.79	23.12	108.67	394	285.33	1.37	4.45	0.34
SLOPE	10	8030	7.33	7.88	5.00	5.92	4.45	0.00	67	67.00	2.57	9.47	0.09
TPI1000	11	8030	0.42	13.93	0.00	0.09	10.38	-109.00	84	193.00	0.03	4.23	0.16
TPI500	12	8030	0.50	9.46	0.00	0.24	5.93	-84.00	73	157.00	-0.08	6.85	0.11
TPI5MI	13	8030	0.96	32.26	2.00	1.73	22.24	-176.00	135	311.00	-0.65	3.48	0.36
TPI90	14	8030	0.12	2.32	0.00	0.01	1.48	-30.00	25	55.00	0.24	24.13	0.03
TwI	15	8030	11.39	9.29	7.00	10.19	2.97	2.00	28	26.00	1.18	-0.51	0.10
RespVarCat*	16	8030	1.27	0.44	1.00	1.21	0.00	1.00	2	1.00	1.05	-0.91	0.00

```

> tuneRF(RMTrain[,-1:-16], RMTrain[,1],
+         stepFactor=.3, plot = TRUE, ntreeTry=300,
+         trace=TRUE, improve=.05)

```

- Run `tuneRF()` function to determine an optimal `mtry` number. The `RMTrain[,1:16]` parameter calls all predictor variables (columns) in the `RMTrain` dataframe except the two response variables. The `RMTrain[,1]` parameter calls the response variable ('`RespVar`') in the `RMTrain` dataframe. You determined these values for your data frame from the `Describe()` output (as exemplified in Figure 34). The `ntreeTry` value (100) should be the `ntree` value you determined from the `plot()` results (Figure 31).

```

tuneRF(RMTrain[,1:16], RMTrain[,1],
       stepFactor=.3, plot = TRUE, ntreeTry=100,
       trace=TRUE,improve=.05)

```



Copy the resulting `mtry` values and the plot to your Model Report.



Enter the optimal `mtry` value on your worksheet.

- Run a second random forest model, called '`tree.rfOpt`' tuned to the optimal '`ntree`' and '`mtry`' parameters. Be sure to edit your `mtry` and `ntree` values based on the results from the previous steps.

```

set.seed(22)
tree.rfOpt<-randomForest(RMTrain$RespVar~.-RespVarCat -RespVar,data=RMTrain, mtry=9,
ntree=300, importance = TRUE, proximity=TRUE)

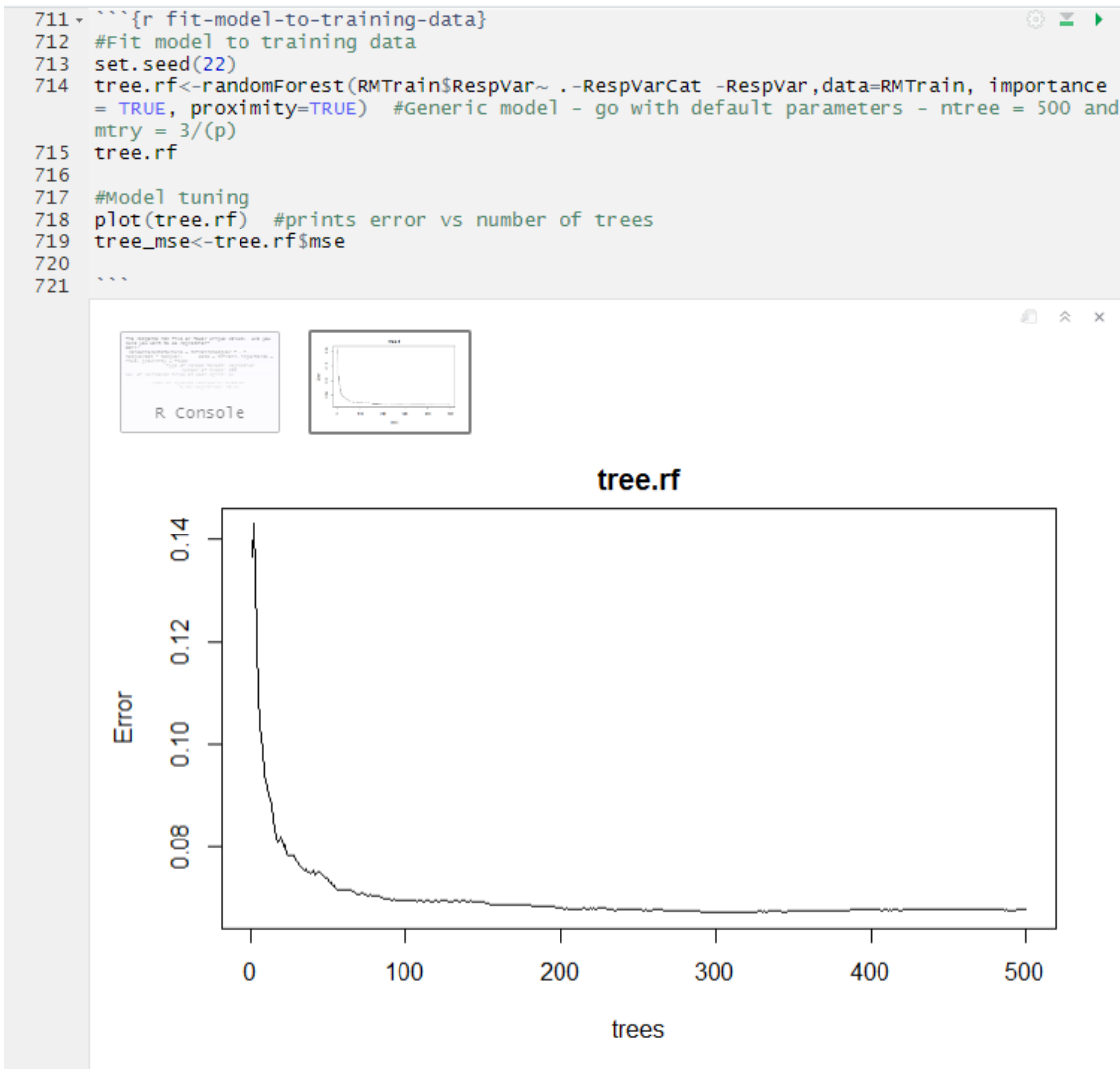
```




Note that you will need to edit the `mtry` and `ntree` values for each optimized model you run.

**Enhanced Script:** The enhanced script will run the `plot(tree.rf)` function in the same chunk of code that fits the first random forest model (Figure 35). The resulting plot will be displayed below the chunk and copied to the `.html` file.

Figure 35: Enhanced Script Code for Plotting tree.rf



To determine an optimal ntree value, discern where the tree.rf asymptote gets closest to the x-axis and use the closest corresponding ODD x-value. In the plot above a value of 301 was selected as the optimal ntree.

 You MUST record your ntree value in the variables.R script (Figure 36), replacing the default ntreeTry value. After editing this value, you must SAVE variables.R.

**Figure 36: Example of ntreeTry value edited in variables.R script**

```
32 # Local variables you will change as you go through the modeling process.
33
34 lform_f_map <- c(17, 30, 43, 44, 45, 53, 56, 59, 66, 68, 77, 87, 97, 38, 52, 54, 61, 62, 67, 69, 71, 73)
35 lform_t_map <- c(65, 29, 93, 49, 49, 84, 29, 93, 49, 29, 84, 49, 93, 84, 84, 93, 83, 84, 65, 49, 29, 49)
36
37 lscape_f_map <- c(27, 19, 26)
38 lscape_t_map <- c(17, 21, 18)
39
40 vegmod_f_map <- c(240, 330, 364, 392, 210, 270, 321, 322, 323, 342, 353, 354, 371, 391)
41 vegmod_t_map <- c(361, 311, 365, 372, 220, 230, 351, 351, 351, 341, 366, 352, 372, 372)
42
43 ntreeTry <- 301
```

After editing and saving variables.R, run the next code chunk (r tune-model) to determine the `mtry` value. The `mtry` values and graph will be added to your .html file. You do not need to put an `mtry` value in the variables.R file. The enhanced script selects the optimal `mtry` value automatically.

Finally, run the code chunk to generate the tuned model. This chunk will also generate the `plot()` function again.

#### Step 4: Assess Model Performance

This section discusses the model performance measures provided by R. The first evaluations of the model are based on the results of the model run itself. These measures are valid for the preliminary models run on training data (75 percent of the total sample) and on the final models run with the entire sample dataset. The next set of evaluations are possible only for the preliminary models run on the training data, as they require an assessment of the performance of the model on the 25 percent of the database set aside as a test population.

##### *Evaluations Based on Random Forest Model*

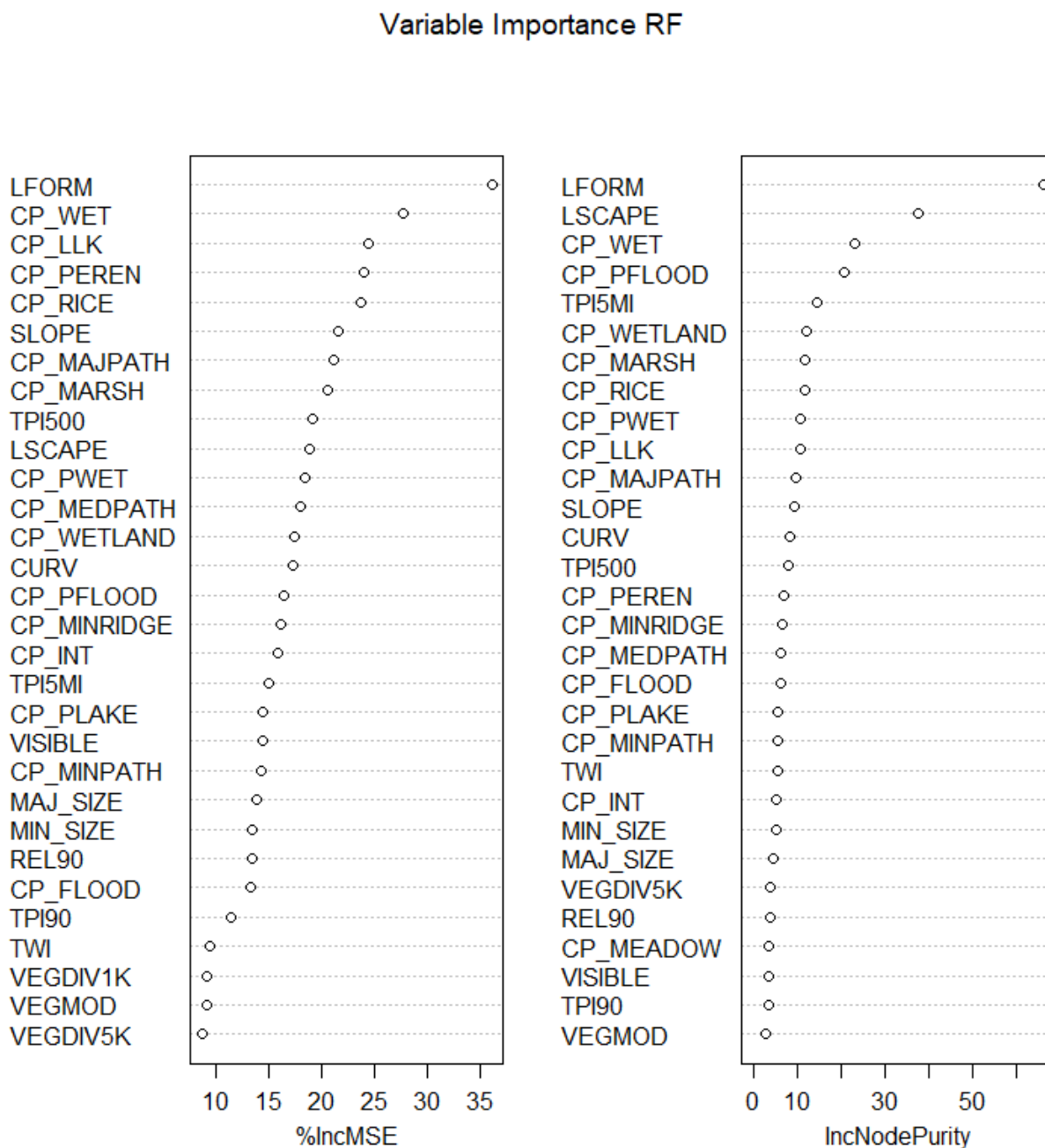
###### Error vs. Number of Trees

You will generate a plot of error vs. the number of trees for your tuned model, similar to the plot you generated for your first model (Figure 31). You will notice, however, that the maximum number of trees on this plot should equal your `ntree` value.

###### Variable Importance

The `randomForest()` function includes a built-in variable importance parameter. The User calls this parameter in the `randomForest()` function using `'importance = TRUE'`. The function `varImpPlot()` will generate a figure showing two plots (Figure 37). The plot on the left shows each variable and the average increase in model error if the variable is permuted (basically stripping the variable of its information). If the variable is useless, you see no or very little change in the '%IncMSE' after permutation. If the variable is useful, you see a big jump in the '%IncMSE' because the information we stripped from that variable (via permutation) is necessary to train a robust model. In Figure 37 it is obvious 'LFORM' (landform) is an important variable for the model and removing it would be costly to overall model accuracy. Conversely, 'VEGDIV5K' (vegetation diversity within five km) is least important and can likely be removed without losing significant model performance. Other variables can likely be removed as well. The plot on the right (Figure 37) represents how well the trees split the data (e.g. node purity). This plot is informative but maybe not as useful to the User as the plot on the left. The `importance()` function will display tabular data quantifying the importance of each variable.

Figure 37: Variable Important RF



**Standard Scripts:**

1. Run the `plot()` function again to plot model error vs. the number of trees for `tree.rfOpt`.

```
plot(tree.rfOpt)
```



Copy this plot to your Model Report.

2. Run `varImpPlot()` to generate the variable importance plots. Also run `importance()` to display tabular data describing the importance of each variable.

```
varImpPlot(tree.rfOpt, main="Variable Importance RF")  
importance(tree.rfOpt)
```



Copy the plots and tabular data to your Model Report.

**Enhanced Script:** The enhanced script produces the graph of error vs. the number of trees in the same chunk of code that runs the tuned model. It then creates the variable importance output at the beginning of the next chunk of code, which proceeds to produce the confusion matrix (below). All of the results will be automatically added to the .html file.

### *Applying the Fitted Model to Test Data*

To ensure model assessment is unbiased, it is important to test the model on sample data not yet observed by the model. In this subsection, we apply the model to the 'RMTest' dataset to assess model performance. All of the measures produced are based on how well the model, derived from 75 percent of the training data, predicts the other 25 percent of the data. We run four models on different 75 percent samples of the data and use the following procedures to evaluate and compare them. We then run a final model based on the entire dataset. We cannot use these procedures to test that model, as there is no test population. However, we can assume that the final model is at least as good as the four preliminary models.

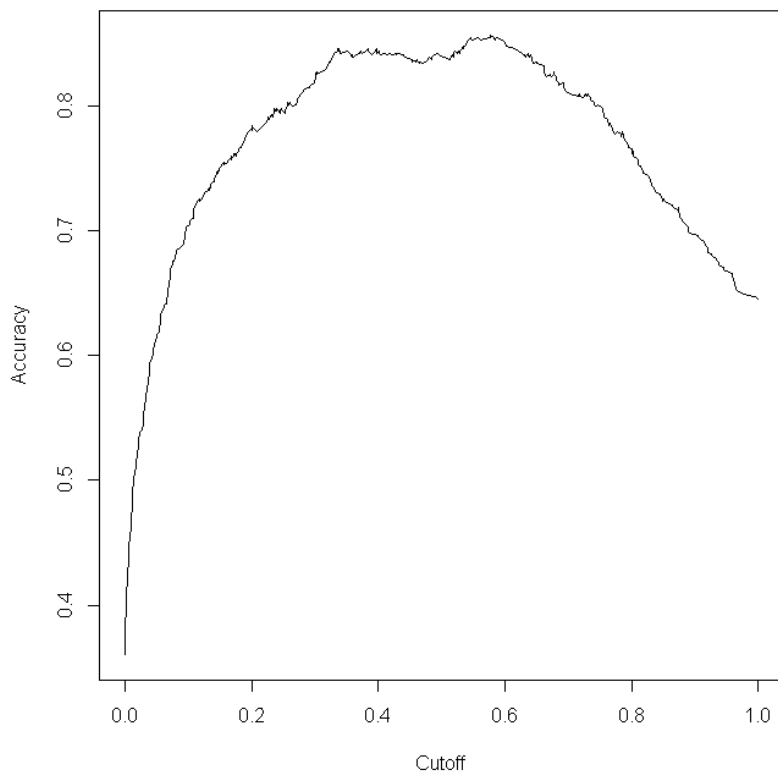
### *Determine 'Optimum' Cutoff Value*

The model output will consist of floating point numbers between 0 and 1. To be useful, the models must be classified to indicate which values indicate high potential for finding archaeological sites (or surveys) and which areas indicate low potential. This requires a decision rule for determining a cutoff value or 'threshold'.

The `ifelse()` function will be used to generate model predictions by applying the fitted model 'tree.rfOpt' to the 'RMTest' dataset. By default, this function uses a conditional statement to assign probabilities above 0.5 to reflect site locations and below 0.5 to reflect background locations. This threshold is **very important** as it defines the sensitivity and specificity of the model; these parameters are discussed below. Oehlert et al. (2007a, 2007b) provides criteria for establishing this threshold relative to these parameters. Since a threshold of 0.5 is arbitrary, we will determine a more meaningful cutoff value before evaluating model performance.

You will generate a plot (Figure 38) of model accuracy versus cutoff values (e.g. probability thresholds). This plot helps you understand overall model accuracy, which does not discriminate between false positive and false negative errors (e.g. treats these errors the same). So, in short, the optimal threshold printed in this output might not be the most ideal threshold for predicting a site or survey, but it will provide more consistent results for comparison of our preliminary models. The optimum value from this curve (maximum accuracy) and its associated cutoff value will also be printed.

**Figure 38: Accuracy vs. Cutoff**



**Standard Scripts:** Run the next set of lines to generate a plot of model accuracy versus cutoff values (e.g. probability thresholds).

```
pred<-predict(tree.rfOpt,RMTest,type='response')
pred<-prediction(pred,RMTest$RespVar)
curve<-performance(pred,"acc")
plot(curve)
```



Copy the results (as shown in Figure 38) to your Model Report.

Now run the following lines to display the maximum overall accuracy and its associated cutoff value:

```
max<-which.max(slot(curve,"y.values")[[1]])
acc<-slot(curve,"y.values")[[1]][max]
cut <-slot(curve,"x.values")[[1]][max]
print(c(Accuracy=acc, Cutoff = cut))
```

You will receive numeric results:

```
> print(c(Accuracy=acc, Cutoff = cut)) #prints  
iated threshold value  
Accuracy Cutoff.292  
0.8559177 0.5763333
```



Record the accuracy and cutoff values on your worksheet.

Run `rm()` to remove the temporary files:

```
rm(curve,max,acc,cut,tree.rf)
```

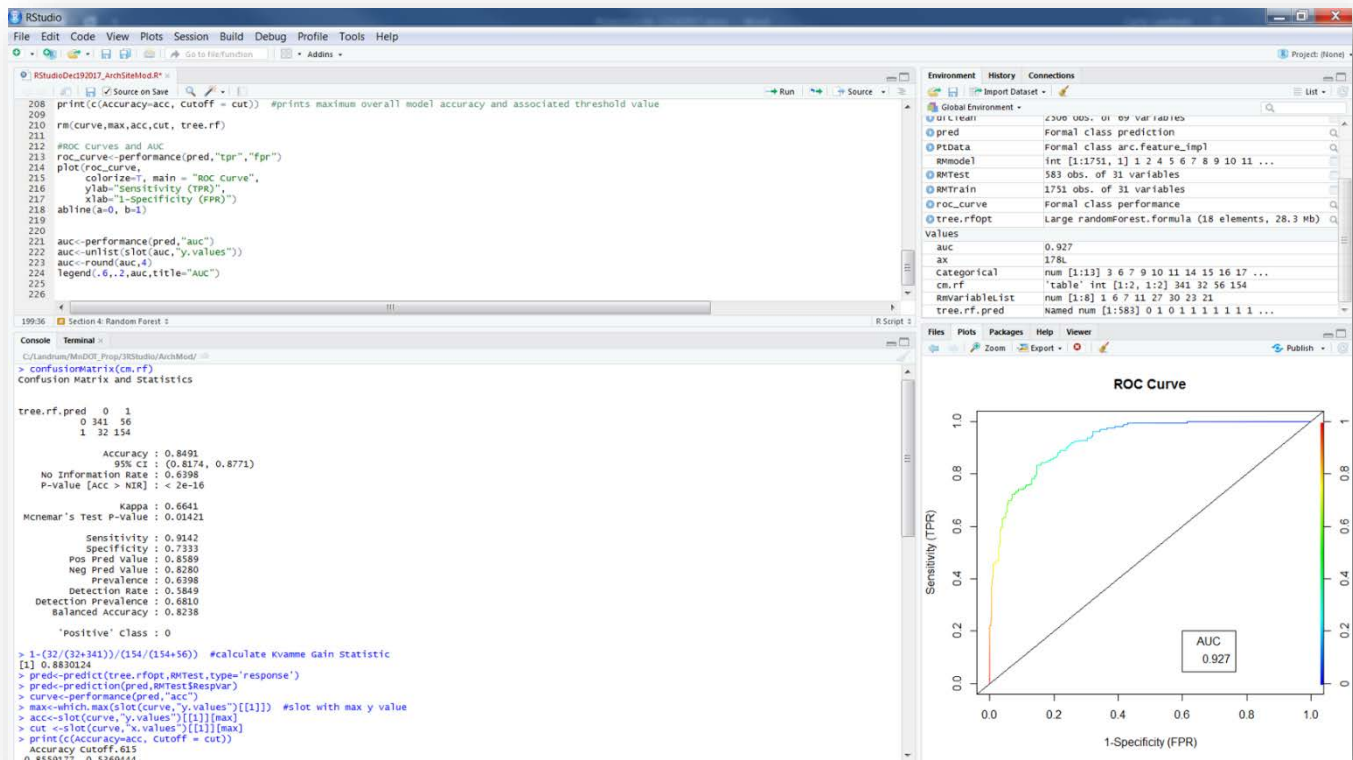
**Enhanced Script:** The enhanced script will run the same code. The .html file will contain the graph and numeric results.

### *ROC Curves*

The ROC curve (Figure 39) plots the sensitivity (y-axis) versus 1-specificity (x-axis). The entire area in this box is equal to 1. The AUC indicates how much space is explained by the model. A perfect model would show an AUC of 1.0. The ROC is colorized according to the threshold probability. This curve will prove useful for establishing a threshold with respect to false positive errors and false negative errors.



Figure 39: ROC Curve



**Standard Scripts:** This section of the script will generate the ROC curve, then calculate the area under the curve (AUC).

```
roc_curve<-performance(pred,"tpr","fpr")
plot(roc_curve,
     colorize=T, main = "ROC Curve",
     ylab="Sensitivity (TPR)",
     xlab="1-Specificity (FPR)")
abline(a=0, b=1)

auc<-performance(pred,"auc")
auc<-unlist(slot(auc,"y.values"))
auc<-round(auc,4)
legend(.6,.2,auc,title="AUC")
rm(RMmodel)
```

The outputs are captured in a graph, as shown in Figure 39. The User can update plot headers by changing the language in quotations (in red above).



Copy the ROC Curve to your Model Report.



Record the AUC on your worksheet.

**Enhanced Script:** The enhanced script will run the same code and add the ROC curve to your .html file.

### Confusion Matrix

The `table()` function will generate a table of predicted values and true values (known site and background locations versus predicted site and background locations). The `confusionMatrix()` function uses this table to generate the confusion matrix and associated performance measures, including overall model accuracy, the Kappa statistic, sensitivity, specificity and more. All of these measures are based on how well the model based on the 75 percent training population predicts the 25 percent test population. The results are illustrated in Figure 40. The matrix itself is explained in Table 5.

**Figure 40: Confusion Matrix and Performance Measures**

```

# Need to reverse the order of columns and rows
> confusionMatrix(cm.rf)
Confusion Matrix and Statistics

tree.rf.pred  1  0
              1 185 79
              0  25 294

      Accuracy : 0.8216
      95% CI   : (0.7881, 0.8519)
No Information Rate : 0.6398
P-value [Acc > NIR] : < 2.2e-16

      Kappa : 0.6336
McNemar's Test P-value : 2.024e-07

      Sensitivity : 0.8810
      Specificity : 0.7882
      Pos Pred Value : 0.7008
      Neg Pred Value : 0.9216
      Prevalence : 0.3602
      Detection Rate : 0.3173
      Detection Prevalence : 0.4528
      Balanced Accuracy : 0.8346

      'Positive' Class : 1
  
```

**Table 5: Confusion Matrix**

Prediction	Reality	
	1	0
1	# Test sites predicted to be sites (True Positives) <b>Sensitivity</b>	# Test background points predicted to be sites (False Positives)
0	# Test sites predicted to be non-site (False Negatives)	# Test background points predicted to be non-site (True Negatives) <b>Specificity</b>

## *Performance Measures*

Oehlert et al. (2007a, 2007b) provides a very detailed overview of the performance measures calculated from the confusion matrix (Figure 40) and their meaning, as does Harris et al. (2015). It is important to note that the accuracy value treats the false positive and false negative errors the same. These errors are not the same when it comes to cost. It is costlier for MnDOT to find a site when the model predicted a background location (false negative) versus discovering a background location when in fact the model predicted a site (false positive). Consequently, threshold values should be set to balance these costs. Oehlert et al. (2007b) recommends that an adequate threshold is one that maximizes specificity for 85 percent sensitivity. That is, the site model should predict 85 percent of sites while maximizing the proportion of background points that are correctly classified as non-sites. This is the sensitivity threshold used in MnModel Phase 3. For MnModel Phase 4, we discovered that we could use a 95 percent threshold while maintaining very high (> 90 percent) specificity values. However, these threshold values must be determined using GIS, and procedures for doing this are discussed in Chapter 3 below. For surveys, false positives are more expensive than false negatives. To assume an area containing a site has already been surveyed and to not survey is more risky than surveying an area that is predicted to have already been surveyed. Thus for the final survey models, we set our threshold to maximum specificity (Chapter 3).

Because R selects the cutoff based on maximum accuracy, without optimizing either sensitivity or specificity, the performance measures provided are weak predictors of how well our final model will perform. In MnModel Phase 4, it is apparent that model specificity is much higher than sensitivity, so the maximum accuracy models are essentially optimized for sensitivity and perform well for surveys. For the site models, these measures are primarily useful for indicating how stable our models are – that is, how much variation we see when we used different subsets of sites and background points to make the predictions.

The performance measures report in R are:

- Accuracy:  $(\text{True Positives} + \text{True Negatives}) / \text{Total Sample}$
- 95% CI: Upper and lower limits of the likely range of accuracy values.
- No Information Rate: Fraction of the landscape that actually does not contain sites. R cannot know this, but bases this estimate on the fraction of the test population that is background points. This is not a useful value, since we set the ratio of sites to background points when we create the background points (Appendix B).
- P-Value (Accuracy > No Information Rate): The probability that model accuracy exceeds the No Information Rate.
- Kappa: A measure of how well the model performed compared to how well it would have performed by chance. Calculation of Kappa also depends on knowing what portion of the landscape actually contains sites, so this is not an accurate measure of model performance.
- McNemar's Test P-Value: McNemar's chi-squared test for symmetry for rows & columns.
- Sensitivity: Percent of sites falling in high probability areas

- Specificity: Percent of background points falling in low probability areas
- Positive Predictive Value: Fraction of locations the rule classifies as sites that actually contain sites. Again, without knowing what portion of the landscape actually contains sites, this measure is meaningless.
- Negative Predictive Value: Fraction of locations the rule classifies as non-sites that are non-sites. Without knowing what portion of the landscape actually contains non-sites, this measure is meaningless.
- Prevalence: Fraction of the landscape that actually contains sites, also known as the *a priori* probability of finding a site. To determine this, we would need to do extensive random archaeological surveys throughout the state.
- Detection Rate: Percentage of the total sample accurately predicted as positive
- Detection Prevalence: Percentage of the total sample predicted to be positive
- Balanced Accuracy: (Sensitivity + Specificity)/2)

### *Kvamme's GAIN Statistic*

The Gain Statistic assesses the balance between the false positive and false negative errors. Harris et al. (2015) provides an overview of the Gain Statistic and references sister reports that elaborate on the use of this statistic for archeological modeling. A Gain Statistic in the rough neighborhood of 0.55 – 0.85 indicates there is good balance in minimizing ‘false negatives at the expense at false positives’ (Harris et al. 2015, pg. 76).

The equation for the GAIN statistic is:

$$1 - (\text{percent area}/\text{percent sites})$$

Where:

- Percent area = the percentage of the total area predicted to be sites/surveys (high/medium probability)
- Percent sites = the percentage of the sites/surveys predicted to be sites/surveys

R can calculate an estimated GAIN statistic for each model using the following formula:

$$1 - (\text{False Positives}/(\text{All Background Points in Sample})) / (\text{True Positives}/\text{All Sites in Sample})$$

Since the background points are not necessarily a reflection of the total area of the region (part of the total area is occupied by site points, for example), this is only an estimate of the Gain Statistic. After we classify the models, we will use GIS tools to measure and re-calculate the GAIN statistic (Chapter 3 below).

## Standard Scripts:

1. Run `ifelse()` function to generate model predictions for the 'RMTest' dataset.

```
tree.rf.pred<-ifelse(predict(tree.rfOpt,newdata=RMTest,type='response')>0.50,1,0)
```

There is no printed output from this function.

2. Run the `tbl()` function to generate a table for the confusion matrix.

```
cm.rf<-table(tree.rf.pred,RMTest$RespVar)[2:1,2:1]
```

3. Run `confusionMatrix()` to generate the confusion matrix and associated outputs.

```
confusionMatrix(cm.rf)
```



Copy these results to your Model Report.



Record the following statistics in your worksheet:

**Enhanced Script:** The enhanced scripts runs the same code to produce the confusion matrix and performance measures. All of these are saved to the .html file.

## Step 5: Complete Four Preliminary Models

Repeat Steps 1-4 a total of four times.

- In Step 1, you must use a different number for your seed values each time to make sure you get a different split of the data into training and testing populations. Add sequential numbers to the RMTrain and RMTest dataframes you create (RMTrain1, RMTrain2, etc.).
- In Step 2, you will also change your seed number prior to running the random forest function.
- In Step 3, you will have new `ntree` and `mtry` values to enter.
- Step 4 should produce different results for each model, and these should be recorded for comparison.

## Step 6: Fit a Final Model to the Entire Dataset

In this Step, you need to fit a random model to the entire dataset. Essentially, your entire dataset becomes the training population. You will skip Step 1, as there will be no test population, and follow Steps 2-4, with some modifications:

- Step 2: You will still need to set a seed so that you can get the same model later if necessary. Use a new number. You will run the model on the ArcModCat dataframe, rather than on RMTrain.

- Step 3: You can optimize your model even without training data. Follow the same procedures as for the preliminary models and record the same information.
- Step 4: The only performance evaluations you will be able to generate in R are the plots of errors vs. the number of trees and variable importance. Record these. The remainder of the performance evaluation for the final model will be done in GIS (Chapter 3).

## Section 5: Generate Region-wide Model Predictions

---

Once the final model is built, the next step is to apply that model to the predictive sample data that was sampled at a 30-m grid mesh resolution (Chapter 1 and Appendix B). This grid has been converted to a text file and is your predictive dataset.

**Standard Scripts:** First, convert the predictive data dataframe (PRED\_NUM) to a dataframe called 'PredMod', then reclassify the categorical variables in 'PredMod' from integer to factor. Use the `str()` function to verify that the factors were correctly defined.

```
PredMod<-PRED_NUM
PredMod[,Categorical]<-data.frame(apply(PRED_NUM[Categorical],2,as.factor))
str(PredMod)
```



Copy the results of the `str()` function to your model report.

Next, remove variables not used for training the model from your predictive data. Use the `psych::describe()` function to verify that these variables were removed.

```
PredMod<-PredMod[,!(names(PredMod) %in% RmVariableList)]
psych::describe(PredMod)
```



Copy the results of the `psych::describe()` function to your model report.

Finally, run the fitted model on the predictive data.

```
tree.rf.pred<-predict(tree.rfOpt,newdata=PredMod)
```

**Enhanced Scripts:** The enhanced scripts will run the same code and list the results in the .html file.

## Section 6: Exporting Model Predictions in .CSV Format

---

**Standard Scripts:** In this section, R will append the model predictions to their respective georeferenced X,Y spatial coordinates included in the prediction grid sample dataset and write the appended table out as a .csv file. This is how we georeference the model predictions for visualization in ArcGIS (Chapter 3).

1. The `cbind()` function binds the model predictions 'tree.rf.pred', generated in the previous step, to the PredMod dataframe, which consists of the sampled prediction grid containing X,Y coordinates.

```
RF_Preds<-cbind(tree.rf.pred,PredMod)
```

2. Next, we write the appended 'RF\_Preds' table out as a .csv file. We use the `write.csv()` function to export the 'RF\_Preds' appended table. Place the directory to which you would like to write the .csv file in quotations. You will also include the name of the .csv file. In this example we save to the C: drive, /MNMODEL4/REGIONS/BGWD/R\_DIR/SITEMOD/ALLSITE1\_V1 folder, and call the file 'ALLSITE1Predictions1.csv'.

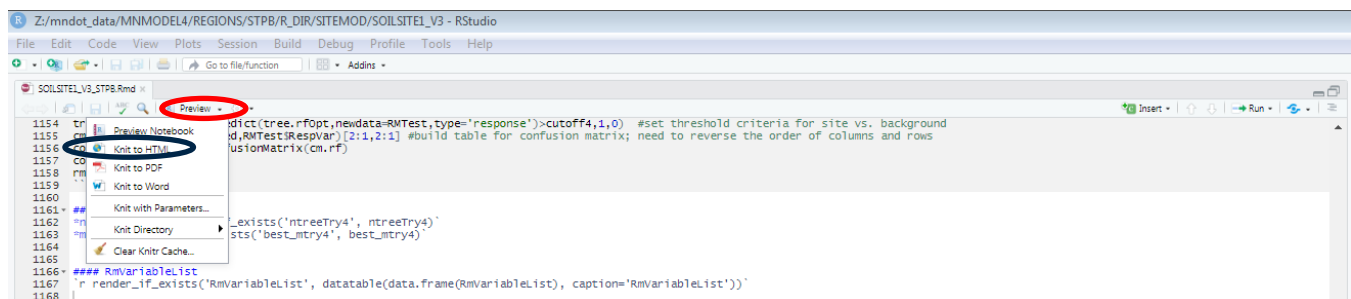
```
write.csv(RF_Preds,"C:/MNMODEL4/REGIONS/BGWD/R_DIR/SITEMOD/ALLSITE1_V1/ALLSITE1Predictions1.csv")
```

**Enhanced Scripts:** The enhanced scripts perform the same functions. The user does not need to specify the output file. The script will be able to tell from the inputs which output file is needed and where it should be stored.

## Section 7: 'Knitting' R Script to .html

After all of the appropriate user inputs have been made to a model's associated variables.R file the user may knit the script to .html to produce a record of the model run.

To do this the user must select 'Knit to HTML' (circled in blue below) from the Preview/Knit dropdown menu (circled in red below).



Knitting will run the entire modeling script from the beginning. Therefore, the user merely needs to run enough of the script to derive, enter, and save all of the appropriate information into the corresponding variables.R file. This information includes the factor level substitutions and the five model `ntreeTry` values. Knitting should produce a modeling report and the appropriate outputs of the modeling script if correct information is entered and saved within the variables.R file that corresponds with the script.

# Chapter 3: Rasterization and Model Classification (ArcGIS)

Rasterization, classification, and model evaluation procedures are all performed in ArcGIS.

## Section 1: Rasterization

---

### Create Floating Point Rasters

The custom tool created to convert the R model output table (in .csv format) to raster is

**TOOLS\TOOLBOXES\MnModel\mnmodel.pyt\MODEL\Convert the csv file output from R to Raster.** This tool will take a .csv file that you created in RStudio and make an ArcGIS grid of the predictive model. The resulting model values are floating point numbers ranging from zero to one. When you have completed models for a region, you should have the following:

- /MNMODEL4/REGIONS/REG/INTERMEDIATE/ARCHMOD\_REG.gdb/ALLSITE: Site probability model for all cells within the region.
- /MNMODEL4/REGIONS/REG/INTERMEDIATE/ARCHMOD\_REG.gdb/SOILSITE: Site probability model for all cells with soils data within the region.
- /MNMODEL4/REGIONS/REG/INTERMEDIATE/ARCHMOD\_REG.gdb/ALLSURV: Survey likelihood model for all cells within the region.
- /MNMODEL4/REGIONS/REG/INTERMEDIATE/ARCHMOD\_REG.gdb/SOILSURV: Survey likelihood model for all cells with soils data within the region.

These floating point rasters are lovely to look at, but difficult to interpret. To be practical, they must be classified into high and low probability areas using some decision rule.

## Section 2: Model Classification

---

Phase 3 models were classified into three probability classes – high, medium and low. When site and survey models were combined to create the survey implementation models, the result was a nine-class model (Table 6).



**Table 6: Classification of the Phase 3 Survey Implementation Model**

<b>CLASS</b>	<b>SITE HIGH PROBABILITY</b>	<b>SITE MEDIUM PROBABILITY</b>	<b>SITE LOW PROBABILITY</b>
<b>SURVEY HIGH LIKELIHOOD</b>	High	Medium	Low
<b>SURVEY MEDIUM LIKELIHOOD</b>	Possibly High	Possibly Medium	Possibly Low
<b>SURVEY LOW LIKELIHOOD</b>	Suspected High	Suspected Medium	Unknown

MnDOT archaeologists concurred that two classes would be sufficient for the site and survey models and that the resulting four-class survey implementation model (Table 7) was more easily interpreted. Thus, for Phase 4 models we needed to select only one threshold, between high and low probability, rather than two (between high and medium and between medium and low). The next question to consider is how best to determine this threshold.

**Table 7: Classification of the Phase 4 Survey Implementation Model**

<b>CLASS</b>	<b>SITE PREDICTED</b>	<b>SITE NOT PREDICTED</b>
<b>SURVEY LIKELY</b>	High Site Potential/Well Surveyed	Low Site Potential/Well Surveyed
<b>SURVEY NOT LIKELY</b>	High Site Potential/Poorly Surveyed	Unknown Site Potential/Poorly Surveyed

For MnModel Phase 4, we conducted an initial round of modeling, evaluated the results, tested different rules for determining our threshold between high and low probability, then refined our modeling procedures and conducted our final round of modeling.

For our initial round of modeling, we developed only one Random Forest model in R, using 75 percent of our population as training data and testing with the other 25 percent. We first classified these initial models using the ‘maximum accuracy’ cutoff values provided by R. These are referred to below as ‘Maximum Accuracy’ models. Since they did not provide the high sensitivity results that we need for archaeological site protection, we then determined cutoff values that insure the inclusion of a specified percentage of sites or surveys within the high probability areas, as was done in Phase 3. These are referred to below as ‘Target Sensitivity’ models. For our final round of site modeling, we created only the 95 percent target sensitivity models.

### **‘Maximum Accuracy’ Models**

The statistical procedures suggest a threshold (cutoff model value) for each model to produce the ‘most accurate’ classification of that model. The stated accuracy is based on both specificity and sensitivity. This accuracy value should be considered an estimate, as it is based on testing the model with only 25 percent of the total sample. The model should be expected to perform somewhat differently for different test populations.

### *Classify Models Using Maximum Accuracy Threshold*

If VALUE > Max Accuracy Threshold, then probability = High

If VALUE < Max Accuracy Threshold, then probability = Low

Run **ClassifyMaxAccuracy** from the CLASSIFY\_MODEL toolbox. The cutoff values for each model should be recorded in your worksheet; you will need to enter them into the tool interface.

This tool will create the following maximum accuracy models:

- ALLSITEMAX
- SOILSITEMAX
- ALLSURVMAX
- SOILSURVMAX

### *Make Composite Maximum Accuracy Models*

We need a composite of the model built on all prediction points but without soil variables (ALLSITEMOD) and the model that used soil variables but is missing values where soils data are absent (SOILSITEMOD). This model will fill in the gaps in SOILSITEMOD with values from ALLSITEMOD. Since the 'maximum accuracy' threshold differs between ALLSITEMOD and SOILSITEMOD, we need to construct this composite (SITEMODMAX) after the two models have been classified. Hence the input models are ALLSITEMAX and SOILSITEMAX. We construct a similar model for surveys.

Run **CompositeMaxAccuracy** from the CLASSIFY\_MODEL toolbox. This tool assumes that ALLSITEMAX, SOILSITEMAX, ALLSURVMAX, and SOILSURVMAX will be found in `\MnMODEL4\REGIONS\REG\INTERMEDIATE\ARCHMOD_REG.gdb`.

This tool will create the following composite maximum accuracy models:

- SITEMODMAX
- SURVMODMAX

Caveat: We would expect that the model that includes soils data would be more accurate, but this is not always the case. It is necessary to evaluate both ALLSITEMAX and SITEMODMAX to see which actually performs better.

### **Target Sensitivity Models**

The archaeologists who are final users of MnModel wanted to see models predicting 85 percent, 90 percent, and 95 percent of sites. Since sensitivity is by definition the percent of sites predicted, the 85 percent sensitivity threshold is that model value whereby 85 percent of the sites are predicted. This may be higher or lower than the maximum accuracy threshold.

We are not concerned about sensitivity thresholds for survey models. We want our survey models to be as accurate and specific as possible. By increasing sensitivity, we risk decreasing specificity and erroneously assuming areas have been adequately surveyed when they have not. All survey implementation models have been built using our initial 'maximum accuracy' survey models.

Note that after the initial round of modeling, we discovered that the 95 percent target sensitivity models worked best for predicting sites. In the final round of modeling, only 95 percent target sensitivity site models were built.

### *Determine Threshold Values*

The procedures described below are for finding the 85 percent sensitivity threshold for the ALLSITEMOD model, but can readily be modified for 90 percent and 95 percent for ALLSITEMOD and all percentages for SOILSITEMOD.

To determine the 85 percent thresholds for ALLSITEMOD and SOILSITEMOD, your unclassified models, attach the model values to the point file containing all of your archaeological sites and background points (\MNMDEL4\REGIONS\REG\SAMPLE\SAMPLE\_REG.gdb\ALLSITE). You do this using the ArcGIS\Spatial Analyst\Extraction tool **Extract Multi Values to Points**.

Once you have sampled ALLSITEMOD and SOILSITEMOD with the ALLSITE point file, open the point file attribute table.

1. You will use simple math to find the following values:
  - $N =$  The number of records produced by the query (SITE = 1 AND ALLSITE IS NOT NULL).
  - $X = 0.85 * N$ . If X is not an integer, round it up
  - $Y = N - X$
2. With (SITE = 1 AND ALLSITE85 IS NOT NULL) selected, sort ascending on ALLSITE. Display only the selected records.
3. Highlight (in yellow) the first Y records in the table (Y records with the lowest values for ALLSITE).
4. The cutoff value will be the ALLSITE value of the record just above your highlighted records, plus some very small number. For example, if the ALLSITE value of the lowest 'blue' record is 0.208013, then the cutoff value will be 0.208012. This will ensure that all records not now highlighted in yellow will fall into the high probability category in the model you classify.
5. Record your cutoff value, then determine and record cutoffs for 90 percent and 95percent ALLSITEMOD and 85 percent, 90 percent, and 95 percent SOILSITEMOD models.

### *Create Target Sensitivity Models*

Use the **TargetSensitivityModels** tool in the **CLASSIFY\_MODELS** toolbox to classify the 80 percent, 90 percent, and 95 percent models for a region and to create the composite SITEMOD85, SITEMOD90, and SITEMOD95 models for evaluation.

This tool will ask you to specify the location of your MNMODEL4 directory, the region abbreviation, and cutoff values for each of the models.

This tool will create the following maximum accuracy models:

- ALLSITE85
- SOILSITE85
- SITEMOD85
- ALLSITE90
- SOILSITE90
- SITEMOD90
- ALLSITE95
- SOILSITE95

- SITEMOD95

If you only wish to create the 95 percent target sensitivity models, use the **TargetSensitivity95Models** tool in the same toolbox.

## Section 3: Model Evaluation

---

It is important to evaluate how well models perform with respect to the entire population of sites or surveys, not just the small test population used to test the models in R. This evaluation is done in ArcGIS. We also use this evaluation process to test the performance of the Phase 3 models with the current archaeological data for comparison to the Phase 4 models. This allows us to measure improvement of Phase 4 over Phase 3.

### Extract Phase 3 Models for Comparison

Since we will be comparing the Phase 4 models to the Phase 3 models, a good first step is to extract the Phase 3 models for the region. Make a new file geodatabase:

`\MNMDEL4\REGIONS\REGION\MODELS\PHASE3_REGION.gdb.`

Use the ArcGIS Spatial Analyst Tools\Extraction **Extract by Mask** tool to extract the following models, using your regions boundary (NOT the buffered boundary) as the mask.

- `\MNMDEL4\STATE\MODELS\Phase3Models.geb\SITEMOD3`
- `\MNMDEL4\STATE\MODELS\Phase3Models.geb\SURVMOD3`
- `\MNMDEL4\STATE\MODELS\Phase3Models.geb\SURVIMP3`

### Sample Models

The first step in the evaluation process is to extract model values for each site or survey and background point.

#### *Sample Site Models with ALLSITE Sample Points*

Use ArcGIS\Spatial Analyst Tools\Extraction tool **Extract Multi Values to Points** to sample all of the models you wish to evaluate. These may include:

- SITEMOD3 (Phase3 site model)
- ALLSITEMAX
- SOILSITEMAX
- SITEMODMAX
- ALLSITE85
- SITEMOD85
- ALLSITE90
- SITEMOD90
- ALLSITE95
- SITEMOD95

#### *Sample Survey Models with ALLSURV Sample Points*

Use Spatial Analyst Tools\Extraction\Extract Multi Values to Points to sample the following grids:

- SURVMOD3 (Phase3 survey model)
- ALLSURV
- SOILSURV
- ALLSURVMAX
- SOILSURVMAX
- SURVMODMAX

## Evaluate 'Maximum Accuracy' Model Performance

A model evaluation worksheet was designed for recording results from the GIS queries and calculating performance measures for each model. To re-create this spreadsheet, use a row for each model and create the columns defined in Table 8.

**Table 8: Column Definitions for Model Evaluation Spreadsheet**

Column	Heading	Definition
A	Model	Name of model
B	Value	The raster value to select from each model to correspond to high or high & medium probability values <sup>1</sup>
C	Site/Survey N	The total number of sites/surveys for which the model does not have NULL values
D	# Sites/Surveys	The number of sites/surveys in the high probability area (e.g. the number of sites/surveys in the selection where the model value is the same as the 'Value' column).
E	% Sites/Surveys	The percentage of sites/surveys predicted by the model $((D/C) * 100)$
F	Background N	The total number of background points for which the model does not have NULL values
G	# Background	The number of background points in the high probability area
H	% Background	The percentage of background points in the high probability area $((G/F) * 100)$
I	Region N	The total number of non-NULL cells in the model raster (from the model's value attribute table)
J	# Cells	The number of cells in the high probability area (from the model's value attribute table)
K	% Region	The percentage of the region classified as high probability $((J/I) * 100)$
L	GAIN	Kvamme's GAIN statistic for the model $(1 - (K/E))$
M	Specificity	The percentage of background points in the low probability area $((F-G)/F) * 100)$
N	ACC	The calculated accuracy value for the model $((D+(F-G))/(C+F))$

<sup>1</sup>For MnModel Phase 4, these values are:

- SITEMOD3 and SURVMOD3: High and Medium probability values (3 and 2)
- All Phase 4 Site Models: High probability = 3
- All Phase 4 Survey Models: High probability = 2

## Section 4: Survey Implementation Model

---

Survey implementation models are models combining site and survey models to provide a better sense of where archaeological field survey is needed.

Survey implementation models can be built from both maximum accuracy site models and target sensitivity site models.

- To make a maximum accuracy survey implementation model, use the best maximum accuracy site model (ALLSITEMAX or SITEMODMAX) and the best maximum accuracy survey model (ALLSURVMAX or SURVMODMAX).
- To make a target sensitivity survey implementation model, use the best target sensitivity site model and the best maximum accuracy survey model.

### Make a Maximum Accuracy Survey Implementation Model

#### *Selection of Best Site Models*

For Site Models, when choosing between two very accurate models we should optimize for sensitivity, the percentage of sites accurately predicted. It is better to call a non-site a site than to call a site a non-site. Thus, when choosing between two models, we need to maximize the number of true positives (the percent of sites in the high probability category).

#### *Selection of Best Survey Models*

Unlike the Site Models, survey models should be optimized for specificity, not sensitivity. It is better to call a surveyed area 'not surveyed' than to call an unsurveyed area 'surveyed.' Thus, when choosing between two models, we need to minimize the number of false positives (the percent of background points in the high probability category).

Select the BEST model of the following pairs:

1. ALLSITEMAX/SITEMODMAX
2. ALLSURVMAX/SURVMODMAX

Factors to consider:

- Model accuracy, as reported by R, is measured based on the model's ability to accurately predict 25 percent of the total sample. If a different 25 percent of the sample is used to test the model, the accuracy is likely to be different that calculated using the GIS data, which looks at the entire database rather than just the 25 percent test population.

- Accuracy values consider both sites (sensitivity) and non-sites (specificity). For site models, go with the model that predicts the most sites even if its calculated accuracy is lower. In other words, the site model should maximize for specificity.
- For the survey model, you want to minimize the number of background points predicted as ‘surveyed’. The survey model should maximize for specificity.
- GAIN is less important. GAIN does not imply accuracy. Also, you can achieve a high GAIN by reducing the amount of land area classified as high probability, even though you may not predict as many sites as a model with lower GAIN.

### *Create SURVIMPMAX*

After selecting the best ‘maximum accuracy’ site and survey models, run **MakeSURVIMPMAX** from the CLASSIFY\_MODEL toolbox. The output will be  
\MNMODEL4\REGIONS\REGION\INTERMEDIATE\ARCHMOD\_REGION.gdb\SURVIMPMAX.

## **Make Target Sensitivity Survey Implementation Models**

### *Select Best Target Sensitivity Site Models*

The best target sensitivity site models will be the ones that predict the highest percentage of sites. These models will not necessarily have the highest accuracy or GAIN values. If the models you are comparing (say ALLSITE95 and SITEMOD95) predict the same number of sites, then select the best model based on the calculated accuracy.

### *Make Target Sensitivity Survey Implementation Models*

You will use the best target sensitivity model for each level and the best ‘maximum accuracy’ survey model to create the survey implementation models for the target sensitivity levels. There are three tools for this in the CLASSIFY MODELS toolbox, one for each sensitivity target level. These are **MakeSURVIMP85**, **MakeSURVIMP90**, and **MakeSURVIMP95**.

## **Section 5: Create Statewide Models**

---

After selecting the best site and survey models and using them to create a survey implementation model, all of the best regional models must be mosaicked into statewide models for distribution to users. For MnModel Phase 4, we elected to use the best 95 percent target sensitivity model from the final model run and the best ‘maximum accuracy’ survey model from the initial model run. We created the following final models:

- SITEMOD4: Statewide Phase 4 site model. This is a mosaic of the best 95 percent target sensitivity site models from each modeling region.
- SURVMOD4: Statewide Phase 4 survey model. This is a mosaic of the best ‘maximum accuracy’ survey models from each modeling region.

- SURVIMP4: Statewide Phase 4 survey implementation model. This is a mosaic of the survey implementation models created from the best 95 percent target sensitivity site models and the best ‘maximum accuracy’ survey models from each modeling region.

## Conclusions

The modeling procedures described in this document worked exceedingly well. Phase 4 models performed much better than Phase 3 models. The results are documented in Hobbs (2019b).

Moreover, the procedures are both robust and flexible. They may be easily modified to customize analysis and modeling for different sets of data and different research goals. All R scripts and GIS tools used for MnModel Phase 4 are available by contacting OES GIS Support, Office of Environmental Stewardship, Minnesota Department of Transportation, MS 620, 395 John Ireland Blvd., St. Paul, MN 55155 ([EnvironmentalDataManager.DOT@state.mn.us](mailto:EnvironmentalDataManager.DOT@state.mn.us)).

## References

Harris, M.D., Kingsley, R.G. and Sewell, A.R.

2015 [Archaeological Predictive Model Set](#). Commonwealth of Pennsylvania Department of Transportation. Final Report. URS.

Hobbs, Elizabeth

2019 [MnModel Phase 4: Project Summary and Statewide Results](#). Minnesota Department of Transportation. St. Paul, MN.

Hobbs, Elizabeth

2019a [Historic Vegetation Model for Minnesota: MnModel Phase 4](#). Minnesota Department of Transportation. St. Paul, MN.

2019b [MnModel Phase 4: Project Summary and Statewide Results](#). Minnesota Department of Transportation. St. Paul, MN.

Hobbs, Elizabeth, Andrew Brown, Alexander Anton, and Luke Burds

2019a [Historic/Prehistoric Hydrographic Models for Minnesota: MnModel Phase 4](#). Minnesota Department of Transportation. St. Paul, MN.

Landrum, Carla and Elizabeth Hobbs

2019 [Vegetation Modeling User’s Guide: MnModel Phase 4](#). Minnesota Department of Transportation. St. Paul, MN.

Oehlert, Gary W. and Shea, Brian

2007a. [User’s Guide for MN/Model Phase 4 S-Plus Software](#). Minnesota Department of Transportation. St. Paul, MN.

2007b. [Statistical Methods for Mn/Model Phase 4](#). Research Services Section, Minnesota Department of Transportation. St. Paul, MN.



## **Appendix A: Software Installation**

## **Appendix B: Data Preparation and Sampling**

## **Appendix C: Tools Handbook**