
Building an R Script Tool

Tutorial Overview

The **R-ArcGIS** bridge is an effective way to make programs created for the R statistical software interoperable with ArcGIS datasets, by providing the `arcgisbinding` package that allows ArcGIS datasets to be read and written easily from the R programming language. Integration of these R programs is most effective when they are implemented as script tools in an ArcGIS toolbox. Doing so makes input/output of datasets and numerical/text parameter values seamless, and enables immediate visualization of feature classes and tables output by your tools. R script tools can be further integrated into larger processes by incorporating them into ArcGIS Model Builder workspaces, creating more advanced geoprocessing workflows.

This tutorial will introduce you to the process of developing R scripts that are ready to be executed as a script tools from an ArcGIS. You will learn how to create and configure new script tools that execute the R scripts in an ArcGIS Toolbox, and use those within ArcGIS Pro or ArcGIS Desktop applications.

Skills

By completing this tutorial, you will become comfortable with the following skills:

- Creating and debugging R scripts designed for use as script tools by ArcGIS applications
- Creating and configuring script tools in an ArcGIS Toolbox to execute R code
- Integrating R script tools into Model Builder workspaces

Time required

The following classroom time is required to complete this tutorial:

- 45 - 60 minutes

Materials Required

- *Technology:*
 - R Statistical Computing Language (version 3.2.0+)
 - RStudio Desktop (version 1.0.153+)
 - ArcGIS Pro 1.1+ (2.0+ recommended) or ArcGIS Desktop 10.3.1+ (10.5.1+ recommended)
 - R-ArcGIS Bindings (1.0.0.128+ recommended)
- *Data:*
 - Data for this tutorial are included as part of the download in the packaged `data` folder.

Data Sources

- City of Toronto:
<https://www1.toronto.ca/wps/portal/contentonly?vgnextoid=1a66e03bb8d1e310VgnVCM10000071d60f89RCRD>
- Toronto Police Service:
<http://data.torontopolice.on.ca/pages/ksi>
- Statistics Canada:
<http://www12.statcan.gc.ca/census-recensement/2016/dp-pd/index-eng.cfm>

Production Date

The Education and Research Group at Esri Canada makes every effort to present accurate and reliable information. The Web sites and URLs used in this tutorial are from sources that were current at the time of production, but are subject to change without notice to Esri Canada.

- Production Date: November 2017

Background Information

The R user community has seen substantial growth in recent years, and is expected to continue growing due to the increasing popularity of statistical data analysis. You can use R in conjunction with ArcGIS to develop tools that can broaden the functionality of ArcGIS. Before you can create these tools, you need to develop an understanding of how to work with the `arcgisbinding` package that is installed by the **R-ArcGIS** bridge. Learning how to incorporate ArcGIS datasets into your R scripts to perform data analysis will greatly extend the possibilities that exist for analyzing both spatial and non-spatial data.

R is different from many other programming languages, in that it was developed primarily for data analysis and computational statistics. While the language and environment have evolved over the past two decades to incorporate many other capabilities, this initial focus provides R with almost every statistical model and data manipulation technique that you would need for modern data analysis. However, it can be a challenge to integrate with other data types or software environments.

The `arcgisbinding` package resolves the challenge of working with ArcGIS datasets by introducing to R a set of functions and classes that facilitate reading, writing, and manipulating ArcGIS datasets in the R programming language. It further enables integration of R with ArcGIS software through the ability to create script tools in ArcGIS toolboxes that execute R code.

By completing this tutorial, you will become familiar with the creation of configuring R script tools in an ArcGIS Toolbox. These R script tools execute functions in R that use the capabilities of the `arcgisbinding` package in combination with the base R language and additional third-party packages. Using this approach, you will be able to enhance the functionality of ArcGIS Pro or ArcGIS Desktop applications with advanced spatial and non-spatial statistical analyses performed by scripts written in the R programming language.

References and Reading

- R-ArcGIS Bridge: full documentation for the `arcgisbinding` package
<https://r-arcgis.github.io/assets/arcgisbinding.pdf>
- R-ArcGIS Bridge: vignette for the `arcgisbinding` package
<https://r-arcgis.github.io/assets/arcgisbinding-vignette.html>
- R-Stats group on GeoNet
<https://geonet.esri.com/groups/rstats>
- R Statistical Computing Language
<http://www.r-project.org/>
- R-ArcGIS Sample Tools on GitHub
<https://github.com/R-ArcGIS/r-sample-tools>
- Coupled Human-Natural Systems (CHANS) Tools
<https://github.com/R-ArcGIS/CHANS-tools>


Part A: Getting Started

To prepare for this workshop, you should have successfully completed the *Getting Started*, *R Scripting Basics*, and *R-ArcGIS Scripting* tutorials. Completing the *Getting Started* tutorial will ensure that you have the required software installed and configured. From the *R Scripting Basics* tutorial, you should be familiar with the **R** and **RStudio** software, and have the basic R scripting skills necessary to complete this tutorial.

In this tutorial, everything discussed within the **R** environment can be completed using the default **RGui** installed with **R**. However, the **RStudio** IDE will be used when referring to working with code in R scripts. In addition, while both **ArcGIS Pro** and **ArcGIS Desktop** support the use of the **R-ArcGIS** bridge, and can execute R script tools from ArcGIS toolboxes, this tutorial will mainly focus on using **ArcGIS Pro** (with significant differences between the two environments noted as appropriate).

Setup

All of the resources required to complete this tutorial will be found in the files included with this tutorial. To get started, complete the following steps:

1. Open the `r-arcgis-rstudio.Rproj` file in **RStudio**
2. Open the *File* tab in the top-right panel (based on the layout options recommended in the *R Scripting Basics* tutorial), navigate into the `R/tools` folder, click on the file `tool-template.R`. The editor pane in **RStudio** should display R source code for a template.
3. In the top-right corner of the source editor, you may click the  maximize icon to toggle the maximized state of the panel, and use as much screen space as possible to display the document's source code.
4. You may open one of the following files depending on which ArcGIS desktop software you are working with:
 - **ArcGIS Pro:** `r-arcgis-pro.aprx`
 - **ArcMap (10.5.x+):** `r-arcgis-arcmap.mxd`
 - **ArcMap (10.3.x=10.4.x):** `r-arcgis-arcmap-103.mxd`
5. You will find that a pre-configured ArcGIS toolbox will be accessible when you have the project (or map document) open:
 - **ArcGIS Pro:** in the catalog pane, an `r-arcgis-pro.tbx` will be listed in the Toolboxes category
 - **ArcMap:** in the catalog pane, the `r-arcgis-arcmap.tbx` can be found in the home directory

Note: refer to "Part D" of the *Getting Started* tutorial if you need to review how to open a project file in **ArcGIS Pro**, a map document in **ArcMap**, and/or how to locate toolboxes within these applications.

Part B: Creating the R Script

The primary component of an R script tool in ArcGIS is the R script. The data processing of the tool will be performed using the R programming language. This enables you to use R to perform tasks inside the ArcGIS environment that might not otherwise be possible. For example, users may want to calculate inferential statistics on the attribute table data associated with a given feature class. Without using a custom tool (e.g., an R tool in ArcGIS), this kind of analysis would typically involve exporting the data from ArcGIS to perform the data analysis in a separate program (e.g., **Microsoft Excel**, **SPSS**, or **R** itself).

An R script must be created before a script tool can be configured in an ArcGIS toolbox that executes your R functions. The `tool-template.R` script opened in the setup steps outlined in *Part A* provides an overview of the main components of a script tool:

1. A function named `tool_exec()` serves as the entry point for your tool. This function is executed when a script tool in an ArcGIS toolbox configured to run your script is run as a geoprocessing task in **ArcGIS Pro** or **ArcGIS Desktop**.
2. The `tool_exec()` function must accept arguments to accept list objects containing input and output parameters that are provided by the script tool when it is executed. By convention, these are usually named `in_params` and `out_params` (though they can be named whatever you prefer).
3. The `tool_exec()` function must return the `out_params` when it completes.

Before returning the output parameters at the end of the tool's execution, the script must do the following.

1. Assign values to output parameters defined as simple string, numeric, or Boolean data types in the configuration of a script tool.
2. Create datasets for each output parameter defined as feature class or tabular. For these, the path to use for each dataset is pre-defined in the value of the output parameter when it is configured as *Required* or *Optional* in the script tool configuration. If the output parameter is configured as *Derived*, then the script must create the corresponding output dataset and then assign the path text to the output parameter value.

The `arcgisbinding` package provides two functions that allow you to set the label and state of the progress bar displayed in the geoprocessing interface:

- `arc.progress_label(<label>)`: Sets the text displayed with the progress bar
- `arc.progress_pos(<pos>)`: Sets the progress bar to display the percent completed (with a value between 0 and 100), or to show continuously animated progress indicator (using a value of -1).

Further information can be reported as messages, warnings or errors, which are displayed appropriately by the geoprocessing interface in **ArcGIS Pro** or **ArcGIS Desktop**. This is done using the following base R functions:

- `print(<text>)` or `message(<text>)`: Displays output as messages in the geoprocessing window.
- `warning(<text>)`: Similar to `print()`, except that text is highlighted in the output, and a warning indicator is displayed with the geoprocessing result.
- `stop(<text>)`: This raises an error with the provided text in R. If the error is not captured in the tool's code, the tool execution in R will stop, the error text will be displayed and highlighted in the geoprocessing messages, and an error indicator is displayed with the failed geoprocessing result.


Unlike the progress bar label, the messages printed by these methods will persist in the geoprocessing results after the tool has completed. This can provide more detailed information for users about the processing taking place during the tool's execution. It is also useful to help troubleshoot while you are developing and testing the R code when it is executed as a script tool by ArcGIS applications.

The steps for creating and configuring a script tool in an ArcGIS toolbox to execute an R script will be outlined in *Part C* of this tutorial. The remainder of this section will focus on how to create and test a new R script in preparation for its use as a script tool in **ArcGIS Pro** or **ArcGIS Desktop**.

Building a sample ANOVA script tool

We will create a sample tool that will accept an input dataset, rank records in quantiles based on a predictor variable, and perform an ANOVA test on a response variable using the quantile ranks as a predictor. The tool will output the ANOVA results, and optionally generate tables with summary statistics and/or the results generated by the ANOVA test.

Start by creating a new script in **RStudio**:

1. In RStudio, click the **Create a New File**  dropdown button and select **R Script**. This will open a new R scripting pane in the top left of the RStudio window.
2. Save the script as `r-arcgis-anova.R` in the `R/tools` folder located in the folder containing the files for this tutorial.
3. Add the following code to define a `tool_exec()` function that accepts two arguments to receive input and output parameters, and returns the output parameters as the result:

```
tool_exec <- function(in_params, out_params) {  
  
  return(out_params)  
  
}
```

Note: in the R script, it is not required to execute `library(arcgisbinding)` or call `arc.check_product()` in order to make use of the functionality provided by the `arcgisbinding` package in the script's code. The package will already be loaded and initialized in the current R session when the tool is executed by **ArcGIS Pro** or **ArcGIS Desktop** from a script tool configured in an ArcGIS toolbox.

3. Insert the following code inside the `tool_exec()`, before the line that returns the output parameters. This code begins by setting the label and position for the progress bar, loading the `dplyr` package, and prints the names of the input/output parameters to the geoprocessing window. When this code is executed by a script tool configured in an ArcGIS toolbox, printing the parameter names can help you identify whether they are in the correct order and/or have the correct names that you expect to use in your code:

```
arc.progress_label('Loading required R packages...')  
arc.progress_pos(0)  
library(dplyr)  
  
print(paste0(  
  'Input parameter names: `', paste(names(in_params), collapse = ``, ` `), `'  
)  
)  
print(paste0(  
  'Output parameter names: `', paste(names(out_params), collapse = ``, ` `), `'  
)  
)
```

Note: This code above requires the `dplyr` package to be installed in your R environment. If you haven't already done this, execute `install.packages('dplyr')` in the R console in **RStudio** to install it from the Internet.

4. Continue the script by inserting the following lines of code. In this part, we are obtaining the values passed to the input parameters that are needed to execute the function, as well as the paths provided for creating two output tables, and assigning them to regular variable names. We are also checking the value of one of the parameters (`predictor_groups`) – if it is not defined, or less than two, our function will default to a value of 2, and print a warning for the user.

```
# Get input/output parameters
in_data <- in_params$in_data
response_var <- in_params$response_var
predictor_var <- in_params$predictor_var
predictor_groups <- in_params$predictor_groups
anova_table <- out_params$anova_table
stats_table <- out_params$stats_table

if (is.null(predictor_groups) || predictor_groups == "NA" ||
    predictor_groups < 2)
{
  warning("Defaulting to minimum of 2 predictor groups.")
  predictor_groups <- 2
}
```

5. Next, insert the following lines of code to update the progress indicator. This section will read the input dataset at the path specified by the `in_data` parameter, and filter it to include only records with non-null values for the two fields specified by the `response_var` and `predictor_var` parameters.

```
# Import data set to data frame
arc.progress_label('Reading data...')
arc.progress_pos(25)

data <- arc.open(in_data)
data_df <- arc.select(data, fields = c(response_var, predictor_var)) %>%
  filter_at(vars(c(response_var, predictor_var)), all_vars(!is.na(.)))
```

6. Continue with the following lines of code to add a `predictor_rank` column containing quantile ranks for the number of quantiles specified by the `predictor_groups` parameter, group the data frame by rank.

```
# Group data into quantiles using predictor field:
grouped_df <- data_df %>%
  mutate(predictor_rank = ntile(data_df[[predictor_var]], predictor_groups)) %>%
  group_by(predictor_rank)
```

7. We can generate a box plot to display the differences in the response variable's distribution for each rank. Without any additional code, this plot will appear to be presented in a new window. It is possible to generate the plot as an image file on disk, and return a path to the file as an output parameter (see the `R/tools/linearK.R` script included with the tutorial files as an example):

```
# Create box plot showing the response variable's distribution for each rank:
boxplot(grouped_df[[response_var]] ~ factor(grouped_df$predictor_rank),
        ylab = response_var,
        xlab = paste(c('Group (', predictor_var, ')'), collapse=''))
```

8. We will expect the output `stats_table` parameter to be optional in a script tool configured to use this function. With that in mind, the following code will check if a path was provided for the output parameter.

If so, a summary table will be calculated with the mean and standard deviation of the response and predictor variables for each group, and written to the output table. Before writing the output, the results will be explicitly passed through the `data.frame()` function to ensure it does not carry any spatial properties (e.g., if the tool is executed with a feature layer or feature class as the `in_data`, instead of a simple table).

```
# Write summary statistics to table
if (!is.null(stats_table) && stats_table != 'NA') {

  # Get summary stats from groups
  arc.progress_label('Calculating summary statistics for each group...')
  arc.progress_pos(50)

  summary_funcs <- c('mean', 'sd')
  summary_df <- grouped_df %>% summarize_at(
    vars(response_var, predictor_var),
    funs_(summary_funcs, args=list(na.rm = TRUE))
  ) %>% data.frame()

  arc.write(stats_table, summary_df)
}
```

9. The next lines will perform the ANOVA analysis, and print the results in as message to be displayed in the geoprocessing results:

```
# Perform ANOVA test
arc.progress_label('Performing ANOVA...')
arc.progress_pos(75)

anova_fit <- aov(grouped_df[[response_var]] ~ factor(grouped_df$predictor_rank))
anova_results <- summary(anova_fit)
print(anova_results)
```

10. If a path is supplied in the value of the optional `anova_table` parameter, then the results of the ANOVA test will be written as a table:

```
# Write ANOVA results to table
if (!is.null(anova_table) && anova_table != 'NA') {
  # Change name of ranking variable label in output, and write to a data frame
  rownames(anova_results[[1]])[1] = predictor_var
  anova_df <- data.frame(anova_results[[1]])

  arc.write(anova_table, anova_df)
}
```


11. Finally, we can update the progress indicator now that the process is done. Also ensure that the output parameters are returned as the result of the function when the process is completed:\

```
arc.progress_label('Done')
arc.progress_pos(100)

return(out_params)
```

Testing your script in RStudio

At this point, your `r-arcgis-anova.R` script should be ready for to be executed by a script tool that you can configure in an ArcGIS toolbox. However, as you develop your scripts, it is more efficient and effective if you can test and debug your script outside of the context of an ArcGIS script tool. In its current form, the script will not do anything on its own. If executed as a standalone R script, it will define the `tool_exec()` function, but it won't actually execute it.

In order to easily test your script, you can add an additional function that simulates executing your `tool_exec()` function from script tool by loading the `arcgisbinding` package, performing any data or workspace preparation, defining two list objects representing input/output parameter values, and then executing the `tool_exec()` function. By placing this code inside another function, it will not interfere with the operation of a script tool that uses it. When you need to test the function outside of the context of a script tool, you can temporarily add a single line at the end of the script that executes your test function (**Note:** you must comment out the line that executes your test before executing it from a script tool in ArcGIS, or it will fail to run).

Append the code below to the end of your script to add a test function, and to execute the test function on the last line of the script:

```
# Function to test in standalone R:
test_tool <- function(){

  # Load the arcgisbinding package...
  library(arcgisbinding)
  arc.check_product()

  temp <- getwd() # Substitute this with another location on disk if you like...
  out_dir = file.path(temp, "data", "ANOVASamples")

  if (dir.exists(out_dir))
  {
    unlink(out_dir, recursive=TRUE)
    Sys.sleep(1) # Wait one second, or the dir.create() method may fail...
  }
  dir.create(out_dir)

  tool_exec(
    in_params = list(
      in_data = "R/tools/CSDsJoined.lyr",
      response_var = "income2016csd.hh_low_income_percent_t",
      predictor_var = "income2016csd.income_median_t",
```



```

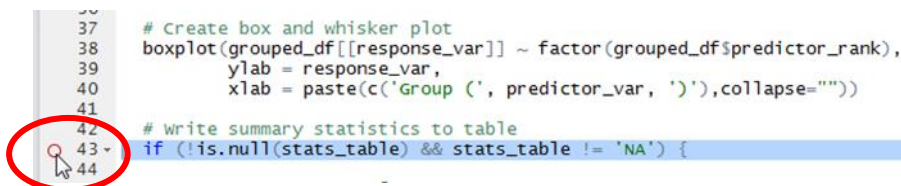
    predictor_groups = 5
  ),
  out_params = list(
    stats_table = file.path(out_dir, "summary_stats.csv"),
    anova_table = file.path(out_dir, "anova_results.csv")
  )
)

print(paste(c("Results saved to: ", out_dir), collapse=""))
}

# Run the test_tool() function
test_tool()

```

In **RStudio**, you can now execute the script as a standalone R script. You can also add break points to aid with debugging during execution. These are added (or removed) by clicking to the left of the line numbers displayed in the left margin of the source code editor pane. Try adding a break point on the `if` statement immediately after the `boxplot()` command:




```

37 # Create box and whisker plot
38 boxplot(grouped_df[[response_var]] ~ factor(grouped_df$predictor_rank),
39         ylab = response_var,
40         xlab = paste(c('Group (', predictor_var, ')'), collapse=""))
41
42 # Write summary statistics to table
43 if (!is.null(stats_table) && stats_table != 'NA') {
44

```

Now you can 'source' the script in the current R session by pressing **CTRL+Shift+S**. **RStudio** will execute the code in the current R session, which will create the `tool_exec()` and `test_tool()` functions, and then run the `test_tool()` function at the end of the script. The tool execution will pause when it reaches the line of code where you enabled the breakpoint. This allows you to inspect variables in the *Environment* tab in **RStudio**, and you can further execute additional commands at the current point in the code (e.g., to analyze or alter parameter values). In this specific example, you may also view the plot that was generated by the `boxplot()` command on the line right before the breakpoint by selecting the *Plots* tab in **RStudio**.

To continue running the code, you can press **Shift+F5**, or click the  **Continue** button that appears at the top of the R console pane in **RStudio**. This will resume execution of the script from the line where the breakpoint was set. It will stop again at any other breakpoints that you set, or when the function is completed.

Remember that you must comment-out the last line that executes your `test_tool()` function before proceeding to use it with a script tool in **ArcGIS Pro** or **ArcGIS Desktop** that you will create in *Part C* of this tutorial. This may be problematic if you forget to comment/uncomment the code when switching between **RStudio** and **ArcGIS** environments. An alternative approach you can use, if you are working with 64-bit R and **ArcGIS Pro**, is to wrap the command inside an `if` statement that only executes the test function if a null value is found for the workspace property in the environment object provided by the `arcgisbinding` package:

```

if (is.null(arc.env()$workspace)) {
  test_tool()
}

```

This will allow you to develop and test your R script both in **RStudio**, and also use it from a script tool executed in **ArcGIS Pro** without the need to comment/uncomment the line that executes your test code. **Note:** *this may cause the script to fail if it is executed in 32-bit R, or used from a script tool in **ArcGIS Desktop** (ArcMap).*

Sourcing external R scripts in your code

You can add extra functions to your script by defining them directly within the script tool, similar to the `test_tool()` function you created to help test the application as a standalone R script. For example, you may want add a function like to the `load_pkgs()` method implemented in the the sample `helper-functions.R` script provided with the tutorial files. This function is helpful to ensure that any missing packages are installed automatically from CRAN when your tool is executed (eliminating the need to ask users to pre-install libraries).

However, it would make sense to source re-usable functions like the `load_pkgs()` function from shared script files. However, when a script tool is executed from an ArcGIS toolbox, the current working directory in the R session that loads your R script does not correspond to the actual file location of the R script on disk, which makes it challenging to source files using relative paths. You can handle this with two different approaches:

- Source additional code by specifying hard-coded absolute paths to scripts in a known location on disk.
- Add an extra parameter to the script tool that represents the path to requires R source files.

The former option is easier, but makes it harder to share your tools with other users, since they must ensure that they either edit the script to change paths, or ensure they place the required files in an identical folder location on their disk. The latter option is more flexible, though it requires an extra parameter to run the tool. You can see an example of this implemented in the `R/tools/random-samples.R` script included with the tutorial files. This tool requires an `R sources` parameter that specifies the path to a folder that would contain required sources (in this case, it must be the `R` folder included with this tutorial files, where the `helper-functions.R` script is located).

Part C: Creating the script tool in an ArcGIS toolbox

The last step for creating an R tool for ArcGIS is to create and configure a script tool in an ArcGIS toolbox. This exposes your R code through the ArcGIS geoprocessing framework, which allows users to execute the code using a GUI that integrates seamlessly with the **ArcGIS Pro** or **ArcMap** interfaces. It also allows your code to be integrated into larger geoprocessing workflows in ArcGIS using Model Builder workspaces.

In this section of the tutorial, you will create and configure a script tool that will execute the `r-arcgis-anova.R` script that you authored during *Part B* of this tutorial (this script should be identical to the `anova-tool.R` provided with the tutorial files – if necessary you can substitute with the provided script instead).

Begin by adding a new script tool the `r-arcgis-pro.tbx` toolbox in **ArcGIS Pro** or to the `r-arcgis-arcmap.tbx` toolbox in **ArcMap**:

1. Open the the `r-arcgis-pro.aprx` project file in **ArcGIS Pro** or the `r-arcgis-arcmap.mxd` map document in **ArcMap** (as outlined in the *Getting Started* section of this tutorial).
2. In the Catalog pane, navigate to the `r-arcgis-pro.tbx` toolbox (**ArcGIS Pro**) or to the `r-arcgis-arcmap.tbx` (**ArcMap**). You may optionally right-click on a folder location, in the catalog pane and create a new toolbox.
3. Right-click on the toolbox, and choose *New > Script...* (**ArcGIS Pro**) or *Add > Script...* (**ArcMap**)

4. In the dialog that appears:

- Enter an alphanumeric name for the tool (no spaces/special characters)
- Enter a descriptive label (the text that is displayed to a user).

In ArcGIS Desktop (ArcMap), you may also enter more detailed information in the tool description in the dialog.

- Recommended: enable the option to store relative paths (so you can package the toolbox with the R script and share it easily with other users)

In ArcGIS Desktop (ArcMap), click 'Next >' to continue.

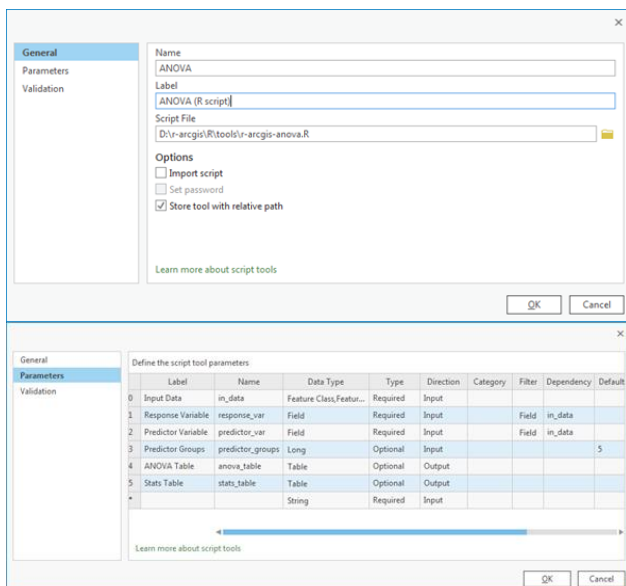
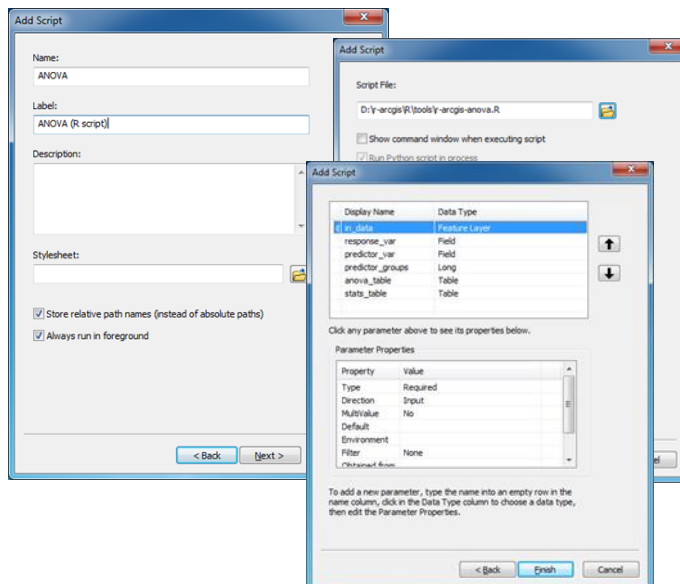
- Specify path to the `r-arcgis-anova.R` script that you created in *Part B* of the tutorial.

5. You now must define the input/output parameters that must be passed to the `tool_exec()` function in your script. In **ArcGIS Pro**, select the *Parameters* section on the left-hand side of the dialog. In **ArcGIS Desktop**, click 'Next >' to proceed to the parameter configuration. Configure the following six parameters:

Name	Label (ArcGIS Pro only)	Data Type	Type	Filter			
in_data	Input Data	Feature Layer *	Required				
response_var	Response Variable	Field	Required	Field**			
predictor_var	Predictor Variable	Field	Required	Field**			
predictor_groups	Predictor Groups	Long	Optional				
anova_table	ANOVA Table	Table	Optional				
stats_table	Stats Table	Table	Optional				

* In **ArcGIS Pro**, multiple data types can be used. For *in_data*, add Feature Class, Table and Table View types

** When the 'Field' filter is chosen, you are prompted for allowed field types. Select short, long, float and double.

*Dialog prompts for creating a script tool in **ArcGIS Pro** (left) and **ArcGIS Desktop** (right)*

In the configuration of your script tool, you may provide user-friendly labels for your parameters in the **ArcGIS Pro** environment. This is not possible in **ArcGIS Desktop**, in which case if you name your parameters with user-friendly text (e.g., 'Input Data' instead of 'in_data'), the script tool will pass parameters with names based on the user-friendly text (where spaces and special characters are replaced with underscores).

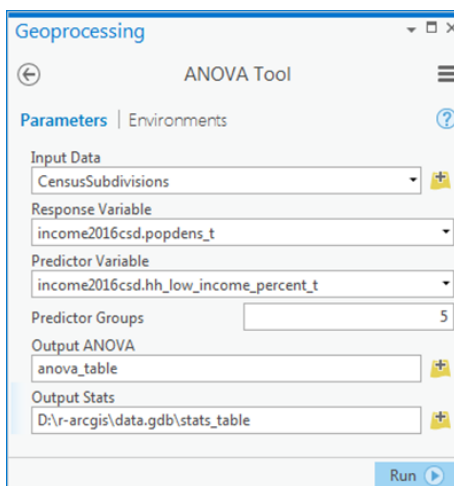
You can learn more about how script tool parameters may be configured from the corresponding documentation:


- **ArcGIS Pro:** http://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/setting-script-tool-parameters.htm
- **ArcGIS Desktop:** <http://desktop.arcgis.com/en/arcmap/latest/analyze/creating-tools/setting-script-tool-parameters.htm>

Now that your script tool has been created and configured in the ArcGIS toolbox, you will see the tool's label listed in the toolbox in the Catalog pane. Double-click on the tool - you will be prompted to define the parameters for your tool. For an input parameter that requires a feature class, feature layer, or table, you will be able to select compatible items in your current project or map document from the dropdown menu. You may also choose any other dataset by typing the full path, or clicking the button to browse for datasets.

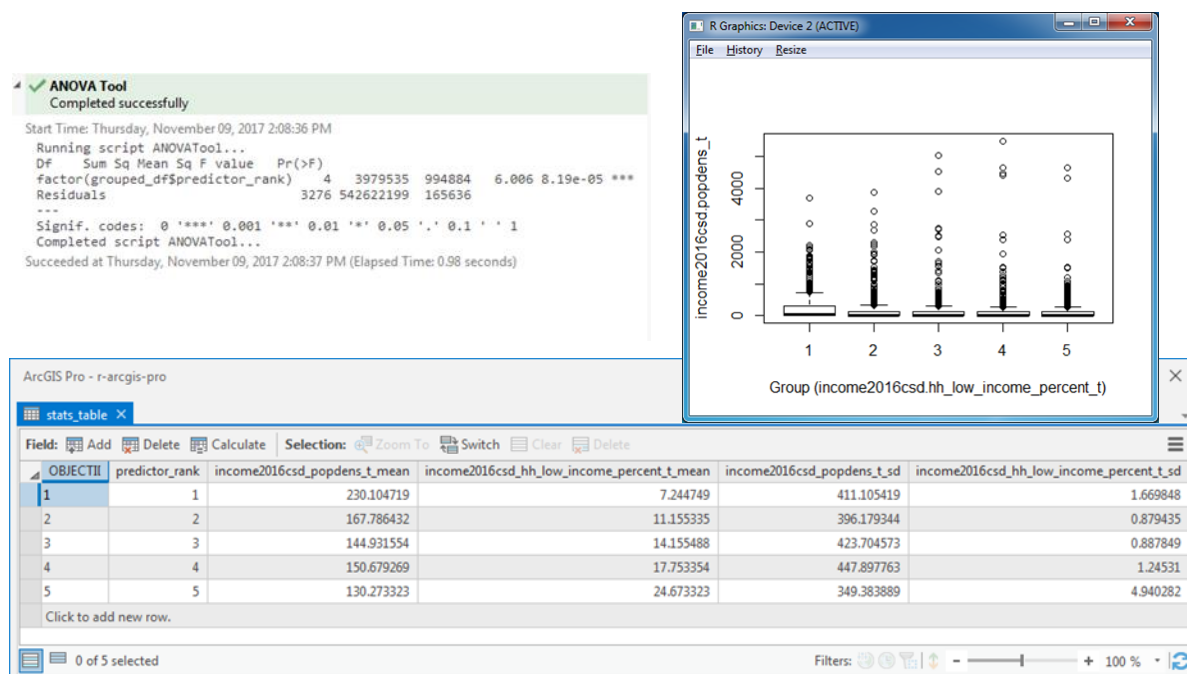
One of the advantages of using Feature Layer and/or Table View data types for input parameters is that you can use datasets that have attributes from multiple sources joined to them, either as in the contents of your current project or map document, or from ArcGIS layer files saved to disk (*.lyr or *.lyrx). **ArcGIS Pro** project file, or the **ArcMap** map document, the CensusSubdivisions layer is pre-configured with a join to the table of data from the 2016 census income statistics. When you select this file as the input dataset, you can choose from any of the joined attributes.

Try executing the tool with the following parameters to test whether population density varies between census subdivisions that are assigned quantile ranks based on percent of households classified as low-income:



You will see the boxplot that is generated by the script, as well as the raw ANOVA results printed in the geoprocessing message window. In addition, the two tables generated by the tool can be found in the Contents pane (in **ArcMap**, click the list by  sources button above the table of contents). Right click on the

`stats_table` that was created, and click 'Open' to view its contents. You will find the mean and standard deviation calculated for the response and predictor variables selected in the analysis above.

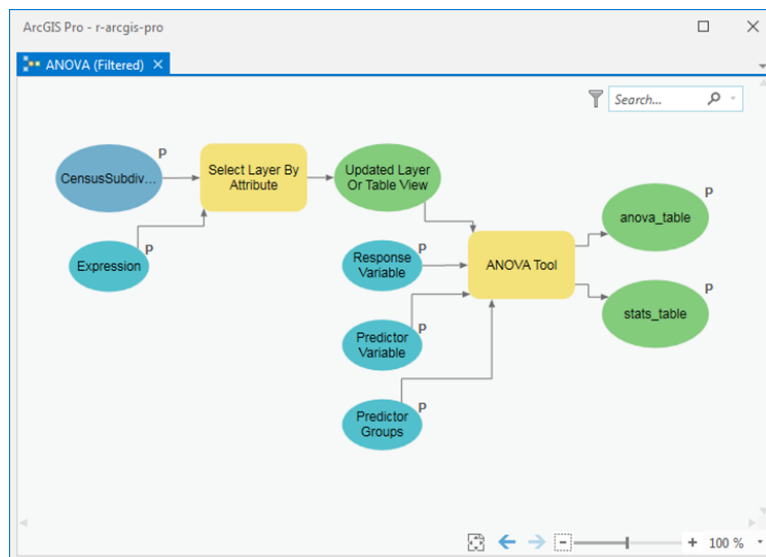


Integrating script tools into larger ArcGIS geoprocessing workflows

You can incorporate your tool into more complex geoprocessing workflows by adding it to a Model Builder workspace:

1. Right-click on the toolbox in the Catalog pane, and choose *New > Model*
2. When the model workspace opens, click and drag your ANOVA script tool into the model workspace.
3. Double-click on the tool in the workspace, and populate the parameters as desired.
4. To expose any of your tool's parameters as variables within the workspace, you can right-click on the tool, and choose *Create Variable > From Parameter > <parameter name>*.
5. To make the variables you exposed as parameter inputs for the Model workspace, right-click on the variable, and enabled the *Parameter* option.
6. Add incorporate any other geoprocessing tools from the ArcGIS toolbox into your workspace as desired.

A sample model has been included in the toolbox provided with this tutorial. Right-click on the *ANOVA (Filtered)* model, and choose *Edit*. You will see an example of the ANOVA script tool, with the *Select Layer By Attribute* geoprocessing task incorporated. This task is configured to prompt for the input dataset, and an expression to be defined by the user. The output is passed to the `in_data` parameter of the ANOVA script tool, with all other parameters for the script tool exposed as parameters for the model workspace.



You can learn more about building model builder workspaces from the corresponding documentation:

- **ArcGIS Pro:** <http://pro.arcgis.com/en/pro-app/help/analysis/geoprocessing/modelbuilder/what-is-modelbuilder-.htm>
- **ArcGIS Desktop:** <http://desktop.arcgis.com/en/arcmap/latest/analyze/modelbuilder/what-is-modelbuilder.htm>

Memory limitations in R

It is important to keep in mind that R loads all of its data objects and performs its processing in the computer's available memory. Depending on the datasets you are using, the types of analyses you perform, and the total RAM available for use by R on your computer, you may encounter limitations.

We can observe this issue with the *Linear K Tool* included in the toolbox provided with this tutorial. This tool requires an input point dataset, and a polyline dataset representing a network that is associated with events described by the point features. Using the `linearK` function available from the `spatstat` package in R, the clustering of points along a network can be measured for a range of scales, and compared to theoretical clustering values that would be expected under the condition of complete spatial randomness. Observed K values that are higher than the theoretical values at each radius suggests clustering, while values below the theoretical K value would suggest dispersion. Multiple simulations can be run to produce an envelope, where values outside this envelope at a given radius could be argued to be significant deviations from complete spatial randomness.

Double-click the tool, and executing it *Linear K* script tool using the KSI and CENTRELINE_WGS84 datasets. These data represent traffic accidents and road centerlines for the entire city of Toronto (**Note: this may run for 5 or so before it fails**). The process in the script will convert the road centerlines into a `spatstat` linear network object, and then use that with the KSI point features to produce a 'point pattern on linear network' object. This is used as the input data for the `linearK` function. When the tool is executed using the complete datasets for all of Toronto, the tool will likely fail with a message as follows:



It is not likely that a PC you are using for this tutorial will be equipped with sufficient RAM to perform this analysis (and likely the process would take a very long time to complete). Thus, it is important to keep these constraints in mind as you develop your R code and use R script tools in ArcGIS.

To see a successful result from the *Linear K* script tool, you can limit the features input into this dataset by selecting a subset of points/lines in both layers for a given area on the map. When the tool is executed, the default behaviour of the `arc.select()` function from the `arcgisbinding` package will limit the features read into the R workspace to only those that are selected on the map.

You may also try executing the *Linear K* tool using the `KSI_byc` and `CENTRELINE_WGS84_byc` feature layers, which are pre-configured subsets of both datasets that include only features within the neighbourhood boundaries of the Bay/Yonge/Church corridor.

Exercise

At this point, you are now able to create and test custom analytical functions in R scripts that use the `arcgisbinding` package from the **R-ArcGIS** bridge, and that are compatible for use as script tools in an ArcGIS toolbox. You have also learned how to create and configure new script tools in an ArcGIS toolbox that define parameters and execute functions in specially-constructed R scripts, and how to incorporate those into larger geoprocessing workflows using Model Builder.

To test your skills, try the following:

- Review the `R/tools/random-samples.R` script, and test it by executing the *Random Samples* script tool that is included in the pre-configured toolbox provided with this tutorial. Once you are familiar with what the script does and how it works, review the documentation for the `spsample()` method provided by the `sp` package in R. Note that there are several types of random sampling methods (controlled by the `type` parameter).
 - Add a parameter to the *Random Samples* script tool to allow a user to select one of the available methods (note, the `Fibonacci` sampling method may not work).
 - Update the `random-samples.R` script to read this additional parameter and provide it as the `type` argument to the `spsample()` method.
- Review the `R/tools/linearK.R` script used by the *Linear K* script tool discussed earlier. Run the tool with the `KSI_byc` and `CENTRELINE_WGS84_byc` datasets included in the project / map documents (i.e., the subsets of traffic accidents and road centerlines for the Bay/Yonge/Church corridor in Toronto).
 - Note that the results from the linear-K analysis suggests clustering is higher than what is expected from the theoretical K value at all radii.
 - Review the point features on the map and the attribute table for the `KSI.shp` dataset in detail in **ArcGIS Pro** or **ArcMap**. You will notice many coincident points at each location (i.e., a point for each party involved in a traffic accident).
 - Update the parameters in the *Linear K* script tool to allow a user to choose a field in the input point features that can be used to identify distinct events.

- Revise the code in the `linearK.R` script to read the new parameter, and use values in the corresponding field to reduce the number of points used in the linear-K analysis to include only distinct events (***Hint:** there is an event identifier code in the KSI datasets...and the `dplyr/spdplyr` packages provide a `distinct()` method that may be helpful*).
- As discussed earlier, this *Linear K* tool cannot function when executed using the complete datasets for all of Toronto. Try incorporating the *Linear K* script tool into a Model builder workspace. In this workspace, add the *Select Layer by Attribute* tool, expose its inputs as parameters for the model, and use its output layer variable to perform a two *Select Layer by Location* processes, one on each of the layers input. The tool should allow a user to use the Toronto neighbourhood boundaries feature layer with a query expression to choose which neighbourhoods to include in the analysis.

Future Considerations

R is a powerful tool with an abundance of resources that can be drawn upon to extend the functionality of ArcGIS. This tutorial has outlined the process of building a basic R script that can be used as a script tool that can be seamlessly integrated into geoprocessing workflows in **ArcGIS Pro** and **ArcGIS Desktop**. You should continue to experiment and create your own R tools for ArcGIS using the procedures you have learned in this tutorial.

<http://hed.esri.ca>

© 2017 Esri Canada. All rights reserved. Trademarks provided under license from Environmental Systems Research Institute Inc. Other product and company names mentioned herein may be trademarks or registered trademarks of their respective owners. Errors and omissions excepted. This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-nc-sa/4.0/).

