# A demographic-based service-area toolkit *

**John M. Brandt** *Yale University*

Transportation distance is a primary indicator of the population served by a given location. For instance, most people would probably drive 5 miles to get to a park or mental health clinic, but many less would choose to drive 50 miles. While some locations, like salons or retail stores, may want to target specific populations, other locations, like WIC centers, hospitals, or parks, aim to serve all populations equally. This toolkit provides an easy means with which to identify groups of people that are better or worse served by any given facility. Using census data, road network data, and the locations of facilities, the toolkit calculates, per county, the percentage of a given demographic within a user-specified driving time or distance from a facility location. The example included in this report calculates the county-by-county access of the total population, black residents, and single mothers to WIC centers in New York. However, this toolkit can calculate the served demographics of any facility type, with any amount of distance, and with respect to any census data (race, income, household type, education, or age). For instance, one could determine whether a youth fashion brand is properly targeting youth, or whether hospitals are properly located near elderly people. Calculating these service percentiles by county allows decision makers to easily identify regions where improvement is needed.

### Introduction

The workflow for this script is broken down into three parts, each a separate ArcToolbox script:

- Creating the service area-layer
- Calculating the percent-coverage of each census block
- Calculating the percent-coverage for a given demographic aggregated to a larger census level

The first part uses a network dataset and the locations of facilities, along with a user-specified service area (in distance or time), to generate a shapefile of the areas served.

The second part uses census block group geometry to calculate the area of the service area within each census block, as well as the area of each census block. These two variables are used to calculate the percentage of each census block that is within the specified distance of a facility.

The third part calculates the percentage of residents of a specified demographic variable within each county in a region that are serviced by a facility. Any census data variable can be used, including race, income, household type, education, and age.

The example included in this report calculates the county-by-county access of the total population, black residents, and single mothers to WIC centers in New York.

*Replication files are available on the author's Github account (http://github.com/johnmbrandt). **Current version**: December 19, 2017; **Corresponding author**: john.brandt@yale.edu.

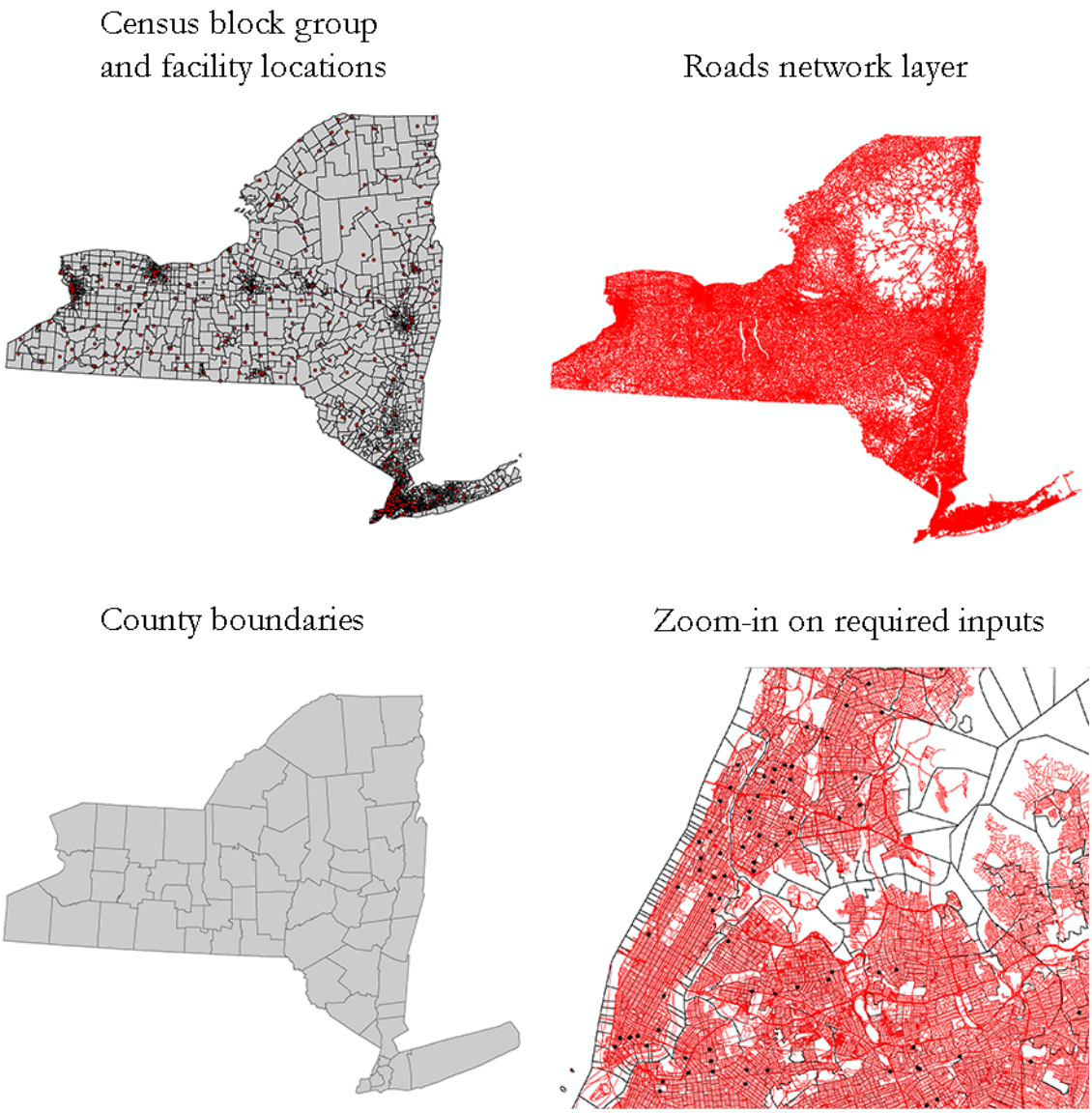


Figure 1: Required inputs

**Part 1 - Network Calculations**

*Inputs*

This script takes the following inputs, in addition to requiring the Network Analyst and Spatial Analyst ArcGIS extensions.

| Input | Description |
|---|---|
| NetworkRoad | Network layer of roads |
| Facility | Feature class containing point location of facilities |
| ServiceAreaRadius | How many feet to calculate service area |
| CensusLayer | Feature class of census block group data |
| DissolveField | Census layer object ID field |
| | |
| outLayerFile | Name of service area layer (.shp) |
| outputDissolved | Dissolved surface area layer (.shp) |
| outputJoined | Final output layer (.shp) |

*Code*

Full code is available online. Begin by importing the required modules, turning the required extensions on, and setting the input and output files and variables. Upon success, we log a message to the console notifying the user that the script is continuing.

```
#Import system modules
import sys, os, math, shutil, arcpy, string, traceback
from arcpy import env
from arcpy.sa import *


try:
    # Check out the necessary extensions
    arcpy.CheckOutExtension("Network")
    arcpy.CheckOutExtension("Spatial")

    # Set environment & output settings
    env.overwriteOutput = True

    # Set input variables
    NetworkRoad = arcpy.GetParameterAsText(0)         # Network dataset (.lyr)
    Facility = arcpy.GetParameterAsText(1)            # Facility locations
    ServiceAreaRadius = arcpy.GetParameterAsText(2)   # Service area radius (feet)
    CensusLayer = arcpy.GetParameterAsText(3)         # Census tracts layer (.shp)
    DissolveField = arcpy.GetParameterAsText(4)       # Identifier in census layer

    # Set output variables
    outLayerFile = arcpy.GetParameterAsText(5)
    outputDissolved = arcpy.GetParameterAsText(6)
```

```
    outputJoined = arcpy.GetParameterAsText(7)

    arcpy.AddMessage('\n' + "The network road input shapefile name is: "
      +NetworkRoad+'\n')
```

Next, the network dataset is built from the input network layer, to which an empty service area layer is appended. Facility locations are added to the network layer, which is then solved. This results in a .lyr output with the polygonal service areas as a sublayer.

```
    # Build the network dataset
    arcpy.na.BuildNetwork(in_network_dataset=NetworkRoad)
    arcpy.AddMessage('\n' + "Network dataset finished")

    # Create a service area layer
    outLayerName = arcpy.na.MakeServiceAreaLayer(in_network_dataset=NetworkRoad,
                                impedance_attribute="Length",
                                travel_from_to="TRAVEL_FROM",
                                default_break_values=ServiceAreaRadius,
                                polygon_type="SIMPLE_POLYS",
                                merge="NO_MERGE",
                                nesting_type="RINGS",
                                line_type="NO_LINES",
                                overlap="OVERLAP",
                                split="NO_SPLIT",
                                excluded_source_name="",
                                accumulate_attribute_name="",
                                UTurn_policy="ALLOW_UTURNS",
                                restriction_attribute_name="Oneway",
                                polygon_trim="TRIM_POLYS",
                                poly_trim_value="250 Meters",
                                lines_source_fields="NO_LINES_SOURCE_FIELDS",
                                hierarchy="NO_HIERARCHY", time_of_day="")

    # Get the output of the service layer
    FacilitiesInterim = outLayerName.getOutput(0)

    # Create a variable referencing the list of sublayers of the service area layer
    naClasses = arcpy.na.GetNAClassNames(FacilitiesInterim)

    # Create a variable storing the name of the sublayer containing the facilities
    FacilitiesLayer = arcpy.na.GetNAClassNames(FacilitiesInterim)["Facilities"]

    # Add the shapefile of locations to the service area layer
    arcpy.na.AddLocations(outLayerName, FacilitiesLayer, Facility, "", "")

    # Solve the service area
    arcpy.na.Solve(outLayerName)
```

In order to do calculations based upon the polygonal area, we must extract the sublayer. To do this, we make use of the arcpy mapping function on the `FacilitiesInterim` object, which contains the output of the service area layer. We use the `naClasses` variable defined above, containing the names of the sublayers, to extract data from the `SAPolygons` sublayer. This layer is copied and saved as a shapefile to the user-specified location.

```
# Use arcpy mapping and the naClasses variable we defined above to create
# a variable storing the polygons sublayer
polygonsSubLayer = arcpy.mapping.ListLayers(FacilitiesInterim,
  naClasses["SAPolygons"])[0]

# Copy the polygons sublayer to the disk, as a shapefile
arcpy.CopyFeatures_management(polygonsSubLayer, outLayerFile)
```

The feature class containing service area polygons has regions of overlap that are served by multiple facilities. The borders of these polygons are dissolved to create a continuous polygon of the service area. This shapefile is also saved to the disk.

```
# Dissolve the borders between service areas within each census polygon
dissolved = arcpy.Dissolve_management(in_features=outLayerFile,
                                      dissolve_field="FromBreak",
                                      statistics_fields="",
                                      multi_part="MULTI_PART",
                                      unsplit_lines="DISSOLVE_LINES")

# Export the dissolved shapefile to the disk
arcpy.CopyFeatures_management(dissolved, outputDissolved)
```

Next, the dissolved service area is intersected with the census block group data. The area of each polygon (representing the service area of each census block group) is calculated, as is the area of the census block group itself. Finally, we join these two layers to create one output shapefile containing block-group census data as well as a field of the serviced area and a field of the total area.

```
# Calculate the intersection between the service area and the census layer
intersected = arcpy.Intersect_analysis([dissolved, CensusLayer],
                                       join_attributes="ALL",
                                       cluster_tolerance="-1 Unknown",
                                       output_type="INPUT")

# Calculate the area of the dissolved polygons within each census polygon
intersected_area = arcpy.AddGeometryAttributes_management(
                                        Input_Features = intersected,
                                        Geometry_Properties="AREA_GEODESIC",
                                        Length_Unit="",
                                        Area_Unit="SQUARE_MILES_US",
                                        Coordinate_System="")
```

```
    # Calculate the area of each census polygon
    Census_area = arcpy.AddGeometryAttributes_management(
                                            Input_Features = CensusLayer,
                                            Geometry_Properties="AREA_GEODESIC",
                                                        Length_Unit="",
                                                        Area_Unit="SQUARE_MILES_US",
                                                        Coordinate_System="")

    # Join the layer of the area of the dissolved polygons to the census layer
    arcpy.SpatialJoin_analysis(target_features=Census_area,
                          join_features=intersected_area,
                          out_feature_class = outputJoined,
                          join_operation="JOIN_ONE_TO_ONE",
                          join_type="KEEP_ALL",
                          match_option="CONTAINS",
                          search_radius="",
                          distance_field_name="")

    # When done, turn off the Spatial Analysis and Network Analysis Extensions
    arcpy.CheckInExtension("Network")
    arcpy.CheckInExtension("Spatial")
```

Report an error message to the arcToolbox script window if the script fails to execute.

```
except Exception as e:
    arcpy.AddError('\n' + "Script failed because: \t\t" + e.message )
    exceptionreport = sys.exc_info()[2]
    arcpy.AddError("at this location: \n\n" +
    traceback.format_tb(exceptionreport)[0]+ "\n")
```

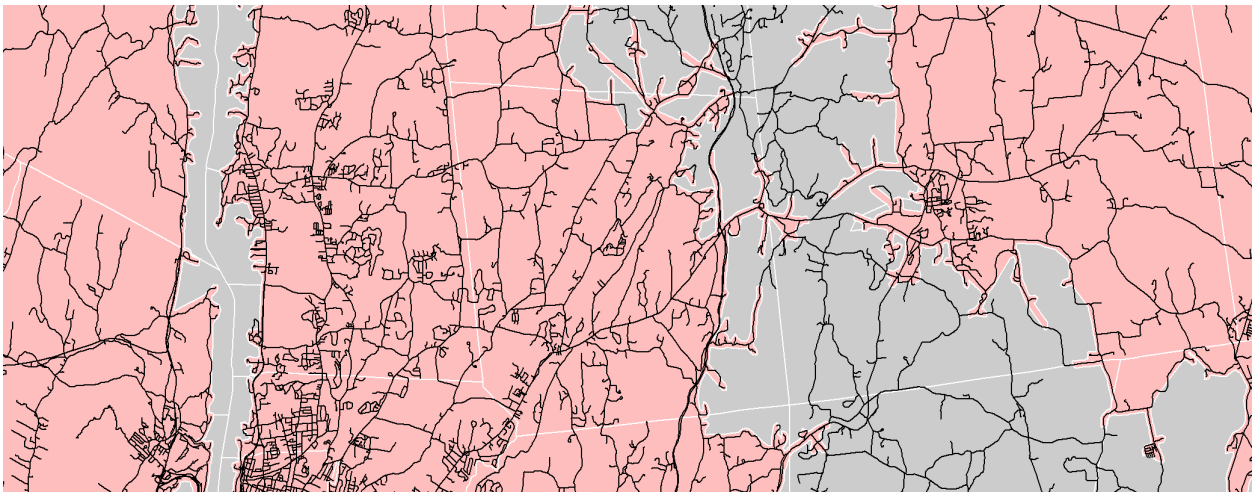

Figure 2: Part 1 output, showing regions within (pink) and outside (grey) the service area.

**Part 2 - Percent serviced area**

*Inputs*

This script takes the following inputs and outputs:

| Input | Description |
|---|---|
| input_shapefile | Output of Part 1, with areas of service per census block group |
| service_area | Field within input_shapefile, areas of service polygons |
| census_area | Field within input_shapefile, areas of census block polygons |
| output_field | Field to be calculated in output_shapefile, percent serviced area |
| output_shapefile | Location of output shapefile |

*Code*

Begin by importing the required modules and setting the input and output files and variables.

```
#Import system modules
import sys, os, math, shutil, arcpy, string, traceback
arcpy.env.overwriteOutput = True

try:

    # Set input variables
    input_shapefile = arcpy.GetParameterAsText(0)
    service_area = arcpy.GetParameterAsText(1)
    census_area = arcpy.GetParameterAsText(2)

    arcpy.AddMessage('\n' + "The input shapefile is" + input_shapefile + ", and"
        + service_area +  " and " + census_area + "are the input fields.")

    # Set output variables
    output_field = arcpy.GetParameterAsText(3)
    output_shapefile = arcpy.GetParameterAsText(4)

    arcpy.AddMessage("The name of the field to be added is " + output_field + "\n")
    arcpy.AddMessage('\n' + "The output shapefile is" + output_shapefile)
```

The input is copied to a new shapefile which will be edited. First, we add a new field to the shapefile that will hold the output data. In order to iterate through each row, we create an `UpdateCursor` object, called `ShpRecords`.

```
    # Create a new copy of the shapefile and save it as the output shapefile
    arcpy.Copy_management(input_shapefile, output_shapefile)
    arcpy.AddMessage('\n' + "Copied")
```

```
    # Add a new field to the output shapefile
    arcpy.AddField_management(output_shapefile, output_field, "DOUBLE", 20, 5)
    arcpy.AddMessage('\n' + "New Field Added")

    # Create a list of iterable records within the output shapefile
    ShpRecords = arcpy.UpdateCursor(output_shapefile)
    arcpy.AddMessage('\n' + "List made")
```

By iterating over each record in the `ShpRecords`, we calculate the percent of area within each census block group that is within the service area of a facility. Specifically, `service_value` obtains the value of the area of the serviced region, and `census_value` obtains the value of the area of the census block group. Dividing the former by the latter results in the percent serviced area, which is saved to the output field defined in the input section.

```
    # Iterate over each index in the shapefile
    for record in ShpRecords:
        service_value = record.getValue(service_area)
        census_value = record.getValue(census_area)
        result = service_value/census_value
        record.setValue(output_field, result)
        ShpRecords.updateRow(record)

    # Delete extraneous temp files
    del ShpRecords
    del record

except Exception as e:
    arcpy.AddError('\n' + "Script failed because: \t\t" + e.message )
    exceptionreport = sys.exc_info()[2]
    arcpy.AddError("at this location: \n\n" +
    traceback.format_tb(exceptionreport)[0]+ "\n")
```

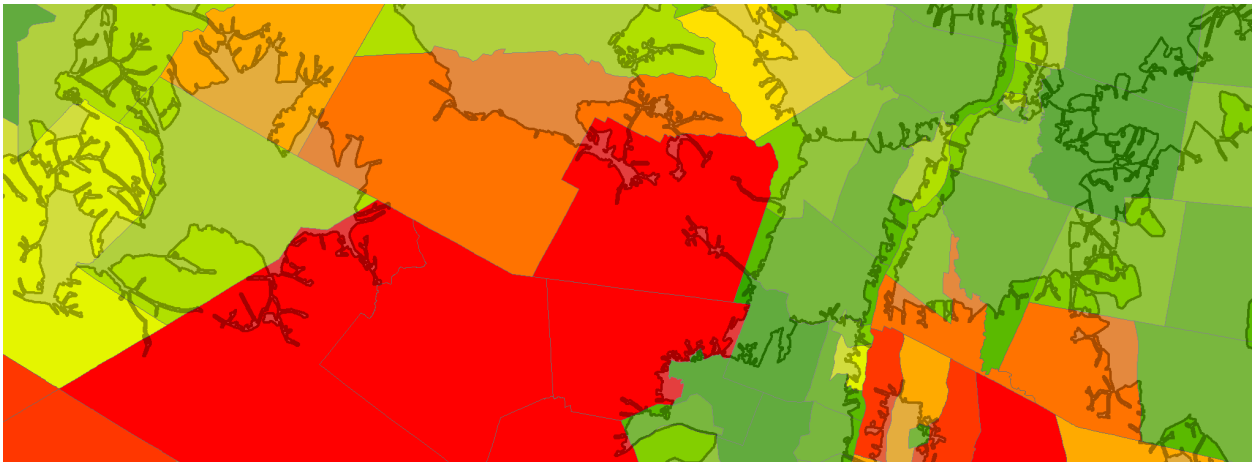

Figure 3: Part 2 output, showing block groups with high (green) and low (red) service coverage.

**Part 3 - Census calculations**

*Inputs*

This script takes the following inputs:

| Input | Description |
|---|---|
| input_part_2 | Output of Part 2, percent coverage of service area by block group |
| input_census_file | Feature class containing county boundaries |
| percent_coverage | Field within input_part_2, percent coverage of service area |
| census_input_field | User-specified field in input_part_2, containing demographic data |
| dissolve_field | Unique identifier within input_census_file to dissolve on |
| total_serve_output | Output field, total number of demographic within county service area |
| percent_serve_output | Output field, percentage of county residents in service area |
| output_shapefile | Location of output shapefile |

*Code*

Using the percent coverage for each census block, a user-defined census variable is multiplied by the percent coverage to calculate the total number of persons of the census variable per census block that are within the service area. This number is aggregated up to the county level, and then divided by the total population of that demographic in the county.

Begin by importing the required modules and setting the input and output files and variables.

```
#Import system modules
import sys, os, math, shutil, arcpy, string, traceback
arcpy.env.overwriteOutput = True

try:

    # Set input files
    input_part_2         = arcpy.GetParameterAsText(0) # input shapefile
    input_census_file    = arcpy.GetParameterAsText(1) # input census file
    arcpy.AddMessage('\n' + "The input shapefiles are" + input_part_2 +
    input_census_file)

    # Set input variables
    percent_coverage     = arcpy.GetParameterAsText(2) # percent coverage
    census_input_field   = arcpy.GetParameterAsText(3) # census input data
    dissolve_field       = arcpy.GetParameterAsText(4) # county FID

    # Set output variables
    total_serve_output   = arcpy.GetParameterAsText(5) # service area * census
```

```
    percent_serve_output = arcpy.GetParameterAsText(6) # percent served
    arcpy.AddMessage("The name of the fields to be added are " + total_serve_output
    + " and " + percent_serve_output + "\n")

    # Set output file
    OutputShapeFile      = arcpy.GetParameterAsText(7)
    arcpy.AddMessage('\n' + "The output shapefile is" + OutputShapeFile)
```

Add a field to the input shapefile that will contain the total number of residents of a demographic variable in each county that are within the service area. As before, create an `UpdateCursor` object of the input shapefile, and then iterate through each record in that shapefile. For each census block group, multiply the value of the census variable by the decimal percent coverage.

```
    # Add field to the input shapefile that will store the total serviced
    # population of input census variable
    arcpy.AddField_management(input_part_2, total_serve_output,
                              "DOUBLE", 20, 5)

    # Create a list of iterable records to loop through
    ShpRecords_calc_number = arcpy.UpdateCursor(input_part_2)

    # Iterate over each index in the shapefile
    for record in ShpRecords_calc_number:
        # get value of user-specified census field
        census_value = record.getValue(census_input_field)
        # get value of percent coverage for census tract
        percent_coverage_value = record.getValue(percent_coverage)
        # calculate total number of serviced population for census tract
        result = census_value * percent_coverage_value
        record.setValue(total_serve_output, result) # save that result
        ShpRecords_calc_number.updateRow(record)

    del ShpRecords_calc_number
    del record
```

Calculate the intersection between the census block group-level and the county-level census boundaries to attach a unique identifier for each county to the census block group data.

```
    # Intersection between input data and input census file, in order
    # to attach county-level unique object identifier
    outPutIntersect = arcpy.Intersect_analysis([input_part_2, input_census_file],
                              join_attributes="ALL", cluster_tolerance="-1 Unknown",
                              output_type="INPUT")
```

Dissolve the census block group data on the county-level unique identifier and calculate the sum of the total and served population of each census block in each county.

```
    # Dissolve the layer on the county boundaries, summing up the served and total
    # populations for each census tract
    dissolved = arcpy.Dissolve_management(in_features=outPutIntersect,
                                out_feature_class=OutputShapeFile,
                                dissolve_field=dissolve_field,
                                statistics_fields=census_input_field +
                                  " SUM;" + total_serve_output + " SUM",
                                multi_part="MULTI_PART",
                                unsplit_lines="DISSOLVE_LINES")
```

Add a new field to the output shape file, and iterate over each county group, calculating the percentage of demographic population within the service area.

```
    # Add a field to contain the percentage of county-wide population
    # served by the facilities
    arcpy.AddField_management(OutputShapeFile, percent_serve_output,
                              "DOUBLE", 20, 5)
    arcpy.AddMessage('\n' + "New Field Added")

    # Create a list of iterable records to loop through
    ShpRecords = arcpy.UpdateCursor(OutputShapeFile)
    arcpy.AddMessage('\n' + "List made")

    # Iterate over each index in the shapefile
    for record in ShpRecords:
        # generate  column for the dissolved serviced population
        # computed when dissolving
        serviced = "SUM_" + total_serve_output[:6]
        # generate column for the dissolved total population
        total = "SUM_" + census_input_field[:6]
        arcpy.AddMessage(serviced)
        # get the value of the serviced population
        serv_value = record.getValue(serviced)
        # get the value of the total population
        tot_value = record.getValue(total)
        # calculate the percent of population served
        result = serv_value/tot_value
        # save that calculation
        record.setValue(percent_serve_output, result)
        ShpRecords.updateRow(record)

    # Get rid of the ShpRecords
    del ShpRecords
    del record

except Exception as e:
    arcpy.AddError('\n' + "Script failed because: \t\t" + e.message )
    exceptionreport = sys.exc_info()[2]
```

```
    arcpy.AddError("at this location: \n\n" +
    traceback.format_tb(exceptionreport)[0]+ "\n")
```

**Output**

Sample outputs can be seen in Figure 5 below. This report calculated the percentage of total, black, and single mother residents in each New York county that are within a 5 mile driving distance of a WIC center. Results are graphed in Figure 4. R functions to generate similar graphs are available in the distributed toolkit online. As can be seen, counties and demographics differ widely in their access to WIC centers, although general coverage for New York is quite good (>80%).

Upstate counties tend to have less access to WIC centers, with Essex (27%) and Hamilton (9%) having the lowest access. Black females in Washington county are 30% less likely to live within the service area of a WIC center than is the average resident. Single mothers have lower access to WIC centers than the general population in only four of 63 counties.

These results are important when considering the future development of WIC centers. In the matter of a few clicks, policy makers can use this tool to realize that Hamilton and Washington counties need to be prioritized for future WIC centers. It also shows that WIC centers are generally serving their target demographic well.

Importantly, this toolkit can be used to calculate the served demographics of any type of facility, with any amount of distance, and with respect to any census data (race, income, household type, education, or age). One could determine whether a youth fashion brand is properly targeting youth, or whether hospitals are properly located near elderly people. Calculating these service percentiles by county allows decision makers to easily identify regions where improvement is needed, and regions upon where to model that improvement.

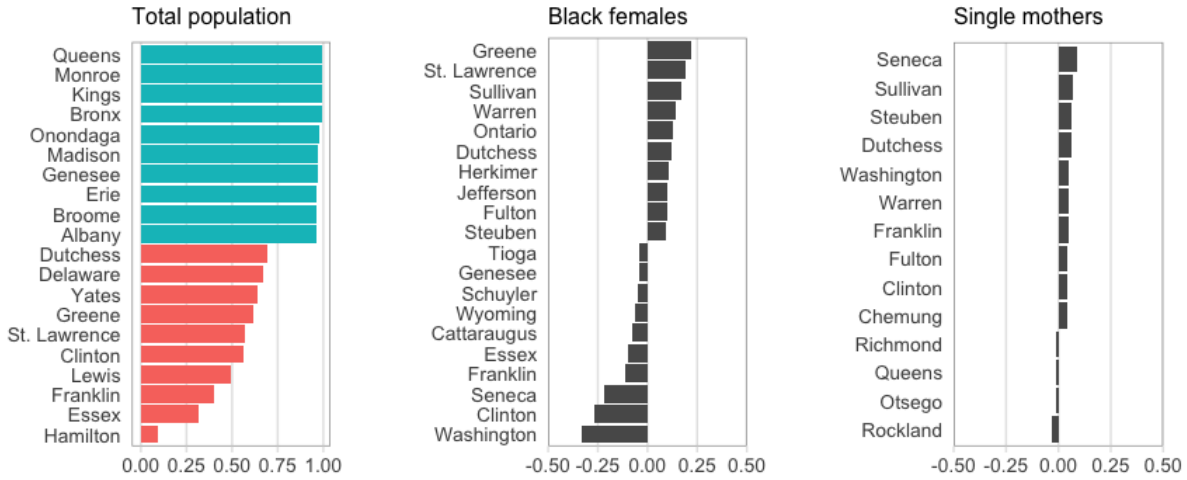


Figure 4: Sample Outputs. **Left)** Top and bottom 10 counties by percentage of population within service area. **Middle)** Top and bottom 10 counties where black females have more (positive) or less (negative) access to WIC centers than the general population. **Right)** Same as middle, but representing single mothers.

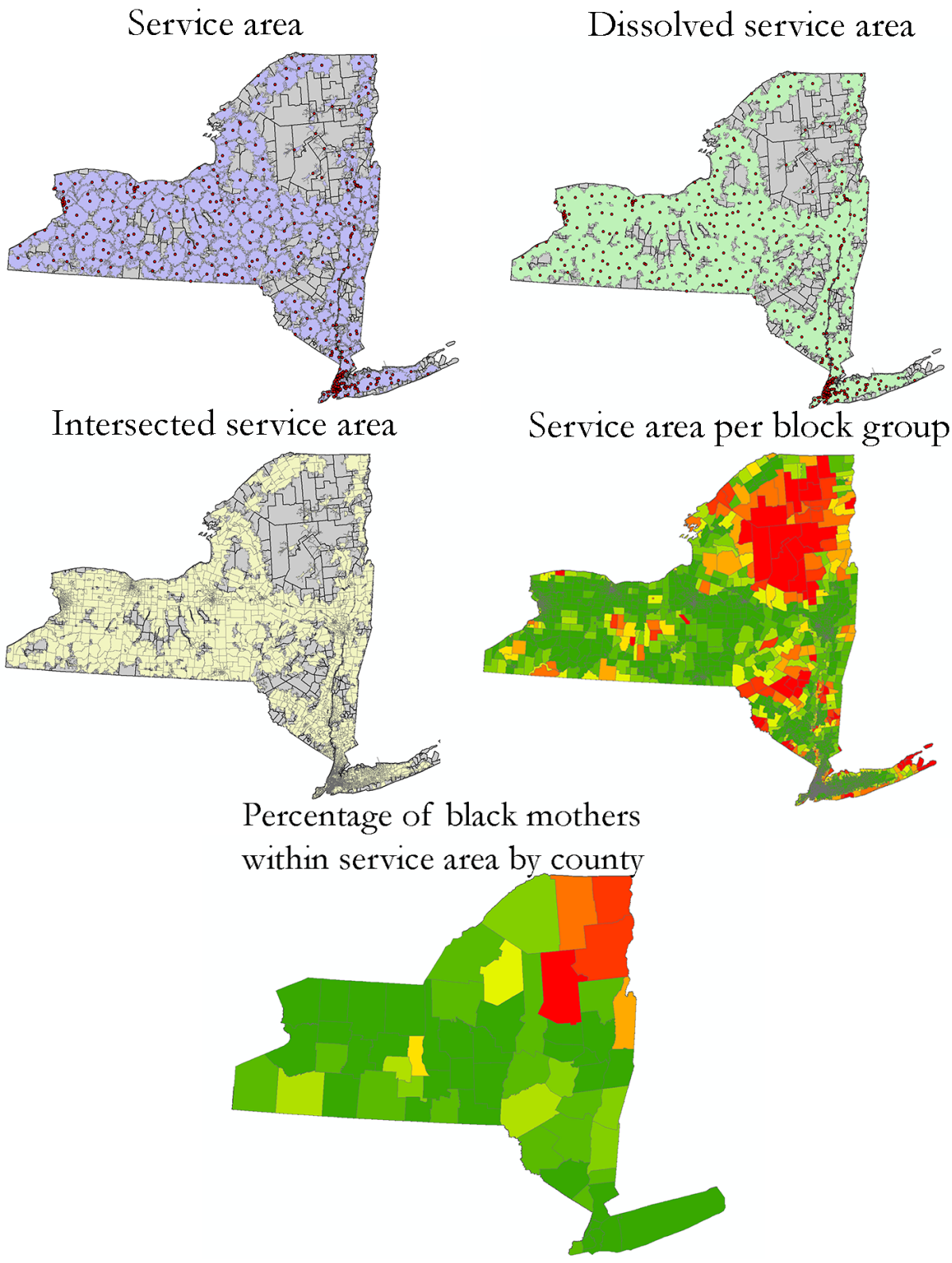


Figure 5: Overview of output files