Data Preparation Documentation

James Jin

Created: 10/24/2025

To prepare all of the data for circuity factor analysis, 4 steps must be completed, 3 of which are scripts. Each of these scripts need to be run in a specific order. For any given study area, each script only needs to be run once.

1. [create\_boundaries.py](#kix.z6pzn9k7aqqy) (optional, will be explained later)
2. [osm\_roads\_planet.py](#kix.oaw6lxo2qlas)
   1. utils.py (do not run this, but contains functions for osm\_roads\_planet.py)
3. [data\_prep.py](#kix.6zwnprhvf41h)
4. [One way functionality](#kix.vqydb79uoubm)

**Purpose and description:**

The purpose of these scripts is to create and prepare all necessary data for circuity factor tools. The basic workflow is to retrieve roads from OpenStreetMaps, merge OSM roads with Forest Service Roads, and process harvest site and sawmill data. The desired end result is to create a harvest site dataset, sawmill dataset, and road network dataset suitable for circuity analysis. Exactly how these work will be expanded on below.

**Data Required:**

There will be a few shapefiles that will be provided with these scripts and should be used for every time any of these scripts are run.

1. Activity\_TimberHarvest.shp - harvest site data for US
2. forisk\_sawmills\_US\_only.shp - sawmill data for US, this data was already filtered only for open sawmills
3. ranger\_districts.shp - all ranger districts in the US, all harvest sites fall within these districts
4. RoadCore\_FS.shp - all FS roads in US

Also included will be a shapefile for physiographic regions. This is not used as an input but will be used to manually extract the desired physiographic region boundaries.

1. physio.shp - physiographic regions

There will be one dataset that must be created/provided by the end user. The boundary shapefile for the study area must be created manually. The physio.shp shapefile can be used to assist in this matter by selecting all desired polygons and exporting to a new shapefile/feature class. It may be in the end user’s best interest to also define the study area by other factors like desired states.

There will also be a few directories that should be manually created to store data and results. When a directory needs to be manually created, the instructions will say.

## 1. create\_boundaries.py (optional)

Description:

This script is very simple and can be done manually instead of desired. This script simply buffers the ranger districts shapefile using the physiographic region, then buffers the resulting shapefile by 125 miles. This script was created for convenience and is considered optional.

There will be one intermediate file that will be created then deleted at the end of the script. After the script finishes, only the output will remain.

**Outputs:**

* Boundary shapefile for sawmills. This will be used to obtain OSM roads.

**Inputs:**

* Park boundaries ([ranger\_districts.shp](#ypn4qy5wm75b))
  + Polygon shapefile
* Physiographic region study area
  + Polygon shapefile/fc
* Output name
  + Can be either a full file path or just the basename + extension
  + The output needs to be a shapefile, it cannot be a feature class in a File GDB. By default, the script puts the output in the same directory as the physiographic region feature class. If this has been converted into a feature class in a File GDB, an output file path must be specified as the third argument.

## 2. osm\_roads\_planet.py

Description:

This script retrieves OSM roads from the US OSM planet file for a bounding box derived from a boundary shapefile. This script will extract roads (not including tracks) in the bounding box and output a feature class in a File GDB.

OSM has several types of roads but not all will be selected. The [types of roads](https://wiki.openstreetmap.org/wiki/Key:highway) used are:

* motorway
* trunk
* primary
* secondary
* tertiary
* residential
* unclassified
* road
* service

Notably, track roads are not included.

This script and the accompanying utils.py script were primarily written by Makiko Shukunobe.

Outputs:

* Road feature class with a ‘oneway’ and ‘reversed’ field from the extracted ‘oneway’ tag
* Log file containing all messages printed

Inputs:

* [OSM planet file](https://planet.osm.org/), downloaded from the Planet OSM website
  + This will be a file ending in .osm.pbf, usually called us-<yymmdd>.osm.pbf. This file is for the entire US, this input should not change unless the file path changes.
* Boundary file
  + Defines the bounding box for obtaining roads. Must be a polygon shapefile. This is obtained from the [create\_boundaries.py](#kix.z6pzn9k7aqqy) script or created manually
* Output name
  + Name of the created feature class. Use only the basename (no extension)
* Directory path
  + Full path for a directory to store the results. Does not need to be empty but is recommended to store all outputs from multiple runs of this script separately.
* Log file path (optional)
  + Full file path and file name for where to store the log. If no option is selected, the default location is the project folder for where the script is in and is called 'osm\_road.log'

## 3. data\_prep.py

Description:

This script does the bulk of the data preparation. Although not listed in any of the steps, all feature classes are projected to the desired spatial reference. This script has 6 main steps:

1. Import road feature class
   1. Use Data Interoperability’s [Quick Import](https://pro.arcgis.com/en/pro-app/3.3/tool-reference/data-interoperability/quick-import.htm) tool to import the road feature class (from [osm\_roads\_planet.py](#kix.oaw6lxo2qlas)). This tool creates a File GDB with the new road feature class inside.
   2. Create a new feature dataset inside the File GDB called ‘Transportation’
   3. This new File GDB will act as the workspace where all temporary files and output files will be stored
      1. This feature dataset will use the spatial reference provided by the user
2. Creating boundary feature classes
   1. Creates a boundary feature class for parks (based on ranger\_districts.shp)
   2. Creates a boundary feature class for sawmills (same as one from [create\_boundaries.py](#kix.z6pzn9k7aqqy))
3. Process harvest site data
   1. Filter for harvest sites within the physiographic region
   2. Filter for year completed to be between 2019 and 2024 (may change depending on when this script is run
   3. Filter out harvest sites below 60 square feet (to remove polygons that were errors)
4. Process sawmill data
   1. Filter for sawmills within the sawmill boundary feature class
5. Merge OSM roads with FS roads - all road outputs will be in the ‘Transportation’ feature dataset
   1. Clip OSM roads with the sawmill boundary feature class
   2. Clip OSM roads with the park boundaries (will be referred to as ‘park roads’)
   3. Merge the park roads with the FS roads
      1. Checks FS roads for duplicity
         1. Duplicates are defined as if 80% or more of the FS road is within 170 feet of the OSM road AND more than 20% of the FS road is farther than 50 feet away.
      2. Export all FS roads that aren’t duplicates
      3. Find points on park roads where the non-duplicate FS roads will snap to
      4. Remove FS roads that won’t snap
      5. Snap the FS roads to the points on park roads
      6. Merge the park roads with FS roads
      7. Use the Integrate and Feature To Line tools to ensure connectivity
   4. Merge the newly merged park and FS roads with the overall OSM roads
      1. Erase any overlapping roads in the overall OSM roads
      2. Merge the result of step 5c with the overall OSM roads with erased sections, result is called ‘complete\_roads’
      3. Use Feature To Line to ensure connectivity
      4. Add a distance field in US miles
6. Create the network dataset from the ‘complete\_roads’ in the ‘Transportation’ feature dataset
   1. The network dataset is called ‘streets\_nd’
   2. IMPORTANT: This network dataset is not ready to use just yet, it needs to be built manually AFTER the one way restriction is manually added.

Notes:

* Must have access to Data Interoperability tools.
* This script creates an instance of a DataPrep class and calls the process() method. However, the user can create their own script that creates a DataPrep object. This allows for custom inputs that lets the user start at any point in the overall process. The process() method contains boolean parameters for each of the steps listed above. Alternatively, each method can be called individually. However, each method uses variables that are modified throughout the entire process. When the user desires to start the process from the middle, the previous methods must be run beforehand and a new DataPrep with new inputs must be created. It has not been tested that the DataPrep class can function like this so proceed with caution.
  + Steps one and two need to be run for other steps to function
  + For all other steps to be run separately, it is recommended to create a DataPrep object with parameters updated to previously created files
  + **Recommendation: run this script as is without skipping any steps**
* The merge process is not perfect. Although most of the network dataset will be functional, there will be some connectivity issues (not enough to affect validity of circuity analysis).

Outputs:

* Boundary files (all projected):
  + Physiographic region
  + Sawmill boundary
  + Park boundary
* Filtered harvest site dataset (step 3)
* Filtered sawmill dataset (step 4)
* Road network dataset (step 5 and 6)

Inputs:

* Workspace
  + Use as the output for the Quick Import tool
  + Will contain all of the outputs
  + Must be a new File GDB, cannot be an existing File GDB
* Total roads feature class (OSM roads)
  + This is the file path for the OSM roads feature class created by the [osm\_roads\_planet.py](#kix.oaw6lxo2qlas) script
* FS roads ([RoadCore\_FS.shp](#ypn4qy5wm75b))
  + This argument should always be the RoadCore\_FS.shp shapefile as it covers the entire US
* sawmill dataset ([forisk\_sawmills\_US\_only.shp](#ypn4qy5wm75b))
  + This argument should always be the forisk\_sawmills\_US\_only.shp shapefile as it covers the entire US.
* harvest site dataset ([Activity\_TimberHarvest.shp](#ypn4qy5wm75b))
  + This argument should always be Activity\_TimberHarvest.shp shapefile as it covers the entire US.
* park boundaries ([ranger\_districts.shp](#ypn4qy5wm75b))
  + The ranger district shapefile will likely be under a different name
  + This shapefile will be the boundaries for all forests containing harvest sites.
  + This is the same shapefile used in create\_boundaries.py
* physiographic shapefile/fc
  + This shapefile/fc should be the desired physiographic region (boundaries for desired harvest sites.)
  + This is the same argument used in create\_boundaries.py
* spatial reference
  + Use the EPSG code for spatial reference

## 4. One way functionality

Description:

1. Oneway functionality must be manually implemented in ArcGIS Pro. To do so, follow these steps:

* Open the streets\_nd network (from [data\_prep.py](#kix.6zwnprhvf41h)) dataset properties in the catalog
* Create a new travel mode called "Driving Distance" by clicking the 3 lines on the top right corner of the properties window and selecting the ‘New’ option.



* + Under costs, ensure length is used for impedance
* Under the ‘Costs’ tab at the top, ensure the ‘distance’ field is used for the evaluators and units are set to ‘Miles’



* Under the ‘Restrictions’ tab, create a new restriction called "Oneway"
  + usage Type: prohibited (-1)
  + under evaluators, for the Along source, set the type to field script and use this value and code block
    - Value=evaluator(!oneway!, !reversed!)
    - Code Block:

def evaluator(oneway, reversed):

return oneway == 1 and reversed == 1



* + under evaluators, for the Against source, set the type to field script and use this value and code block (same as the above image except for the reversed value in the code block)
    - Value=evaluator(!oneway!, !reversed!)
    - Code Block:

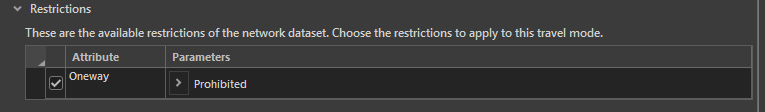
def evaluator(oneway, reversed):

return oneway == 1 and reversed == 0

* Restriction page should look like:



* Go back to the ‘Travel Modes’ tab and make sure the Oneway restriction is checked



2. Build the network using the Build Network tool

## Pseudocode

This section will go over the workflow for the road merging process in [data\_prep.py](#kix.6zwnprhvf41h) that is done over the course of three functions:

1. prep\_roads()
2. merge\_park\_roads()
3. create\_road\_fc()

DEF prep\_roads(self):

SET the output path to a feature class inside Transportation feature dataset

Project on the total roads data to the output path

Clip the projected total roads data using sawmill boundaries

SET class variable total roads to newly clipped roads

Clip the park roads using the park boundaries, with output to class variable

park roads

Project on the FS roads data

SET class variable FS roads to newly projected FS roads

ENDDEF

DEF merge\_park\_roads(self):

Filter the FS roads for intersecting the park boundaries

SET class variable FS roads to filtered FS roads

Generate points along each FS road at 4% intervals with end points included

CALL Near() on each point to park roads with a search radius of 170 feet

CALCULATE new field (IS\_NEAR) for points if it’s close to a park road using 1s

and 0s

CALL Near() on each point to park roads with a search radius of 50 feet

CALCULATE new field (VERY\_NEAR) for points if it's very close to a park road

using 1s and 0s

SET near\_dict to an empty dictionary

SET very\_near\_dict to an empty dictionary

CALL AddField() to create a ‘duplicate’ field to the FS roads

SET sc to a search cursor on points with fields ORIG\_FID, IS\_NEAR, VERY\_NEAR

SET uc to an update cursor on FS roads with fields OBJECTID and DUPLICATE

FOR each row in sc:

IF ORIG\_FID is not in near\_dict:

SET near\_dict[ORIG\_FID] = IS\_NEAR

ELSE:

SET near\_dict[ORIG\_FID] += IS\_NEAR

ENDIF

IF ORIG\_FID is not in very\_near\_dict:

SET near\_dict[ORIG\_FID] = VERY\_NEAR

ELSE:

SET near\_dict[ORIG\_FID] += VERY\_NEAR

ENDIF

ENDFOR

FOR each row in uc:

IF near\_dict[OBJECTID] > 20 and very\_near\_dict[OBJECTID] > 5

SET DUPLICATE = 1

ELSE:

SET DUPLICATE = 0

ENDIF

CALL updateRow on row

ENDFOR

Export all FS roads that are not flagged as duplicates

SET class variable FS roads to newly exported FS roads

Generate end points for each FS road

CALL Near() on the end points to park roads with search radius of 100 feet and

output distance in miles

Create a new point feature class called road\_points

SET sc to a search cursor for the end points with fields NEAR\_X and NEAR\_Y

SET ic to an insert cursor for road\_points with field SHAPE@

FOR each row in sc:

Create a new point with NEAR\_X and NEAR\_Y

IF NEAR\_X and NEAR\_Y are not -1:

CALL insertRow into road\_points with the new point

ENDIF

ENDFOR

SET remove\_count to 1

WHILE remove\_count > 0:

SET remove\_count to 0

Make a feature layer from the end points that are not close to park roads

Make a spatial join between end points and FS roads with intersection

Make a feature layer of end points with join count >= 2

SET orig\_fid\_dict to an empty dictionary

SET sc to search cursor on end points feature layer with field ORIG\_FID

FOR row in sc:

IF ORIG\_FID is in near\_dict:

SET orig\_fid\_dict [ORIG\_FID] += IS\_NEAR

ELSE:

SET orig\_fid\_dict [ORIG\_FID] = IS\_NEAR

ENDIF

ENDFOR

SET sc to search cursor on end points feature layer with field ORIG\_FID

FOR row in sc:

IF orig\_fid\_dict[ORIG\_FID] exists and equals 2:

orig\_fid\_dict[ORIG\_FID] -= 1

ENDIF

ENDFOR

SET uc to an update cursor on FS roads with field OBJECTID

FOR row in uc:

If orig\_fid\_dict[OBJECTID] exists and == 2:

CALL deleteRow()

INCREMENT remove\_count by 1

ENDIF

ENDFOR

ENDWHILE

CALL Snap() on NFS roads to the road\_points fc vertices with 100 feet radius

CALL Merge() on the park roads and FS roads, set the output to “all\_roads”

CALL Integrate() on “all\_roads” with cluster tolerance of 0.5 feet

CALL FeatureToLine() on “all\_roads” with output to “osm\_nfs\_combined”

SET class variable combined roads to “osm\_nfs\_combined”

ENDDEF

DEF create\_road\_fc(self):

SET workspace to transportation feature dataset

CALL Erase() on the total roads using combined roads

CALL Merge() on the erased roads with the combined roads

CALL FeatureToLine() on the merged roads to get completed roads

CALL RepairGeometry() on the completed roads

CALCULATE a new distance field for completed roads in US miles