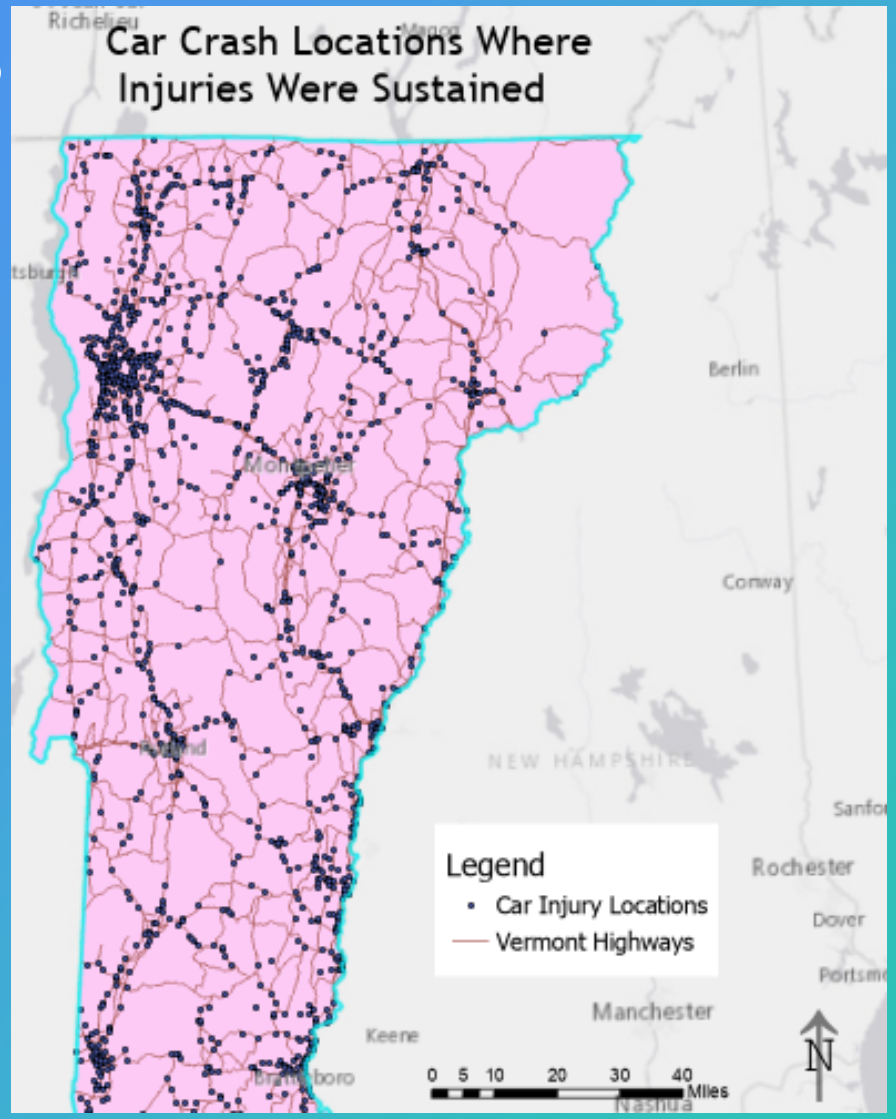
VERMONT TRAUMA CENTER – CAR CRASH ANALYSIS

Kyle McCarthy

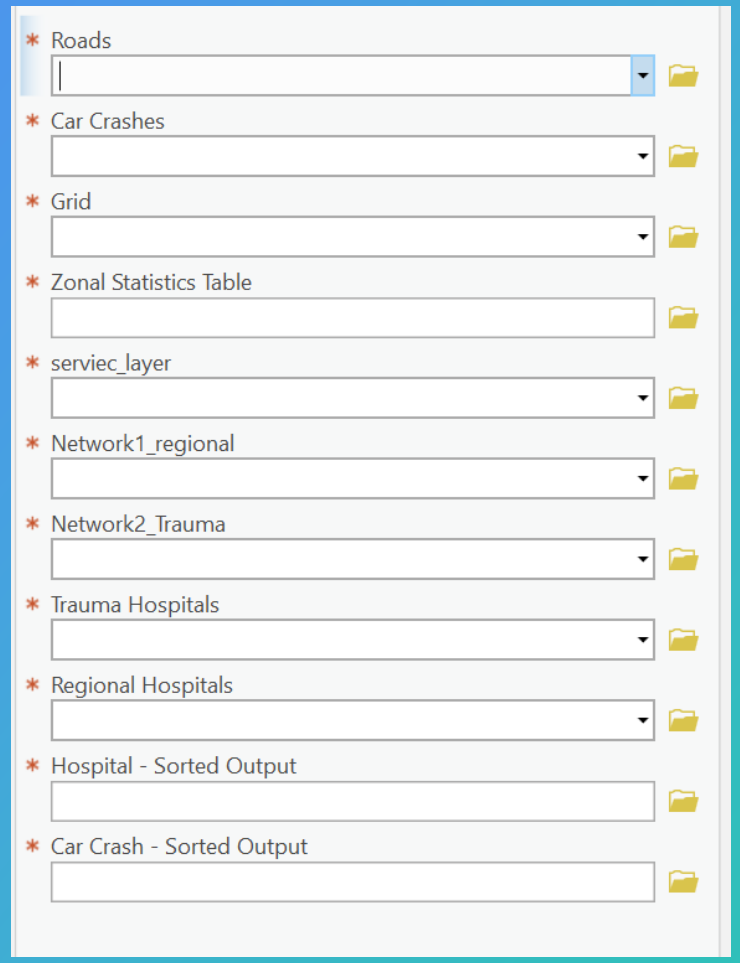
LARP 743

Introduction



In rural regions where a hospital is not nearby, car crashes can significantly risk death. Often, deciding whether a person survives a car crash is their proximity to the nearest trauma centers. In Vermont, a very rural state, there are only 2 nearby trauma centers; the University of Vermont Medical Center, and Mary Hitchcock Hospital at Dartmouth, New Hampshire. This analysis asks where an additional trauma center can be best to reduce the risk of car crash-related deaths in Vermont? Utilizing ArcPy, I develop a script to determine where a new trauma center is best suited in the state. The analysis incorporates a variety of GIS techniques ranging from raster analysis to network analysis to data management.

Parameters



INPUTS

- Road Shapefile
- Car Crash Locations
- Grid (fishnet)
- 1 Service Area Network Analysis Layer
- 2 Closest Facility Network Analysis Layers
- Trauma Hospital Locations
- All Regional Hospital Locations

Outputs

- Table Summarizing car crash density score
- A Ranking of each hospital, indicating where a trauma center should be placed
- A Ranking of identified car crash locations, indicating which common car crash location is most underserved

Manipulating the Road Shapefile

Create fields for min and max X, Y coordinates representing the end of each road feature class, along with a feature class where azimuth is calculated

Geometries calculated for X min, X max, Y min, Y max

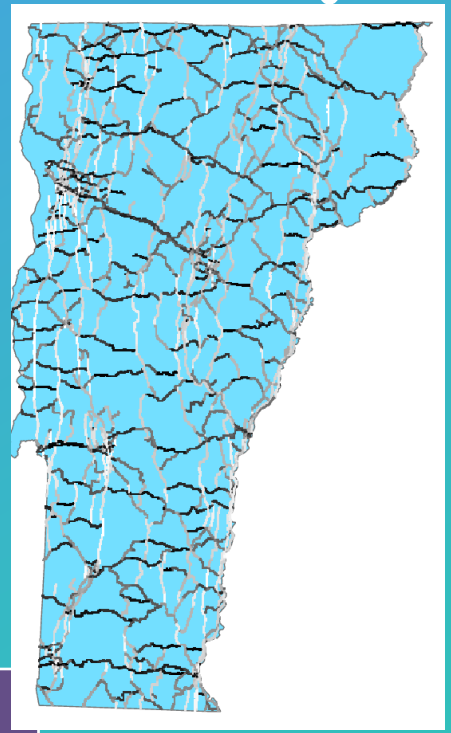
The azimuth (bearing direction) of each road was calculated, yielding a value between 0 and 90

Road Feature Class was converted to raster layer

Raster widened through focalc statistics median

This value was then divided by 1000000000 to minimize the value, so that the road direction is accounted for, but insignificant in a future calculation

Direction Raster Output



Point Density of Car Crashes

- Utilizing arcgis' point density function, the density of car crashes was calculated for each location
- These values were reclassified to be represented as integers, utilizing the following code:
  - $Abs(INT(\ln(plus(PointDensityValue, 0.0000001))))$
- These values were reclassified to represent integers ranging from 1-5
- The resulting value from the Direction Calculation was added to this result

SHIFTING RASTER CELLS

N- S Shifting

(1, 1)	(0,2)	(-1,0)
(0, 0)	(0,0)	(0, 0)
(-1, 0)	(0, -2)	(-1, -1)

E- W Shifting

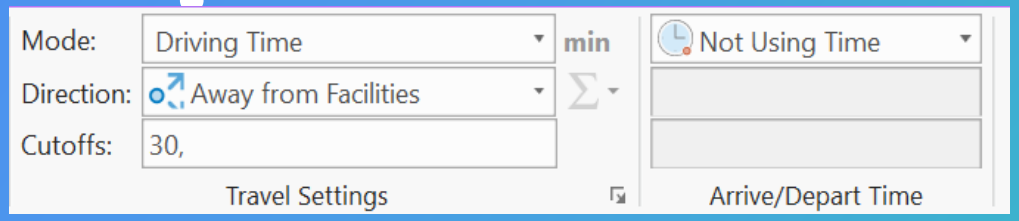
(-1, -1)	(0,0)	(0, -1)
(-2, 0)	(0,0)	(2, 0)
(0, 1)	(0 , 0)	(1, 1)

Everything Else

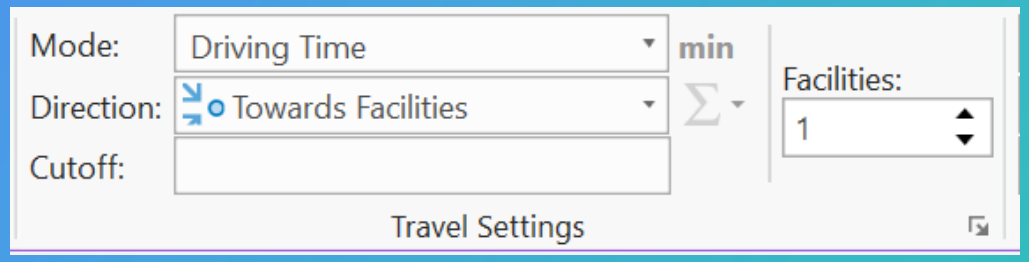
(-1, -1)	(-1,0)	(-1,1)
(0, -1)	(0,0)	(0, 1)
(1, -1)	(1, 0)	(1, 1)

The next set of code attempts to ensure that the higher values of car crash densities are located on roads. This section (line 94) accounts for the direction of the road, since we want to shift the cells along roadways to best account for car crashes. In this code, the movement of the pixels is dependent on the road direction. To accomplish this, there is a series of if-else statements asking for the cell value – int(cell value) * 1000000000. If it is less than 30, it trends E-W. If it is greater than 60 it trends N-S. The code takes the mean of all surrounding pixels. Since the cells are being shifted in the direction of the road, it is expected that pixels in the direction of the road will have higher values. This is necessary because car crashes do not occur in areas where there is not a road. All other pixels that do not have a direction associated with it, or do not trend N-S or E-W take the mean of all values with its neighborhood. The code ensures that outliers are not included in the output, which is when the raster value is below a mean specified standard deviation from neighboring values. After trial and error, a SD limit of 1 best maximized these high car crash locations.

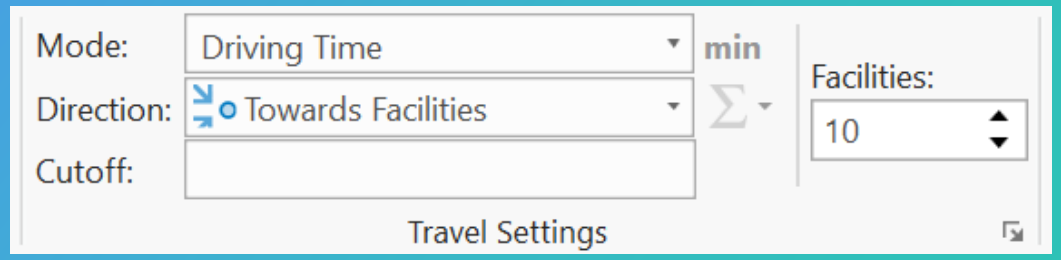
Service Area Inputs



Trauma Center Closest Facility Inputs



Regional Hospital Closest Facility Inputs



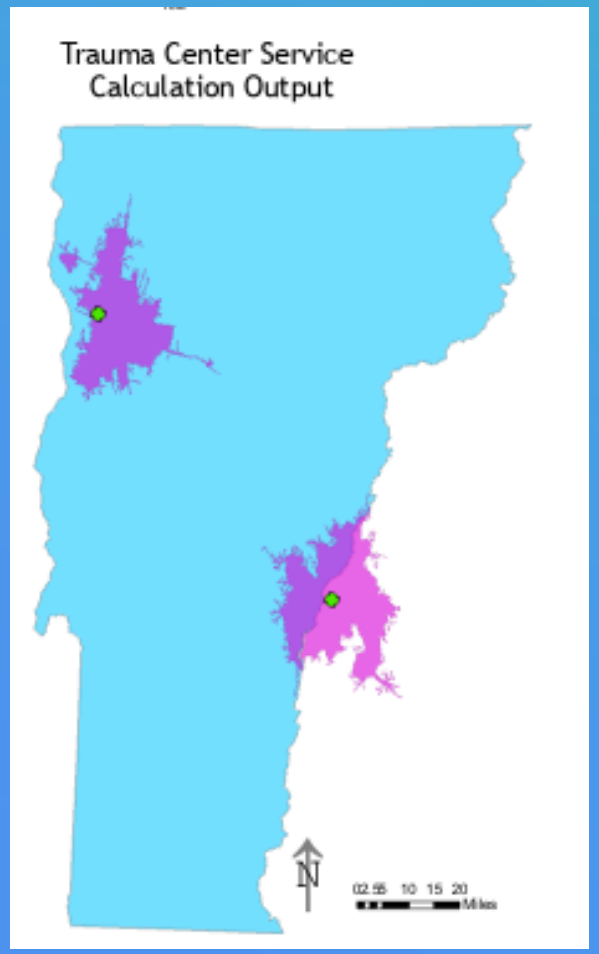
Network Analysis

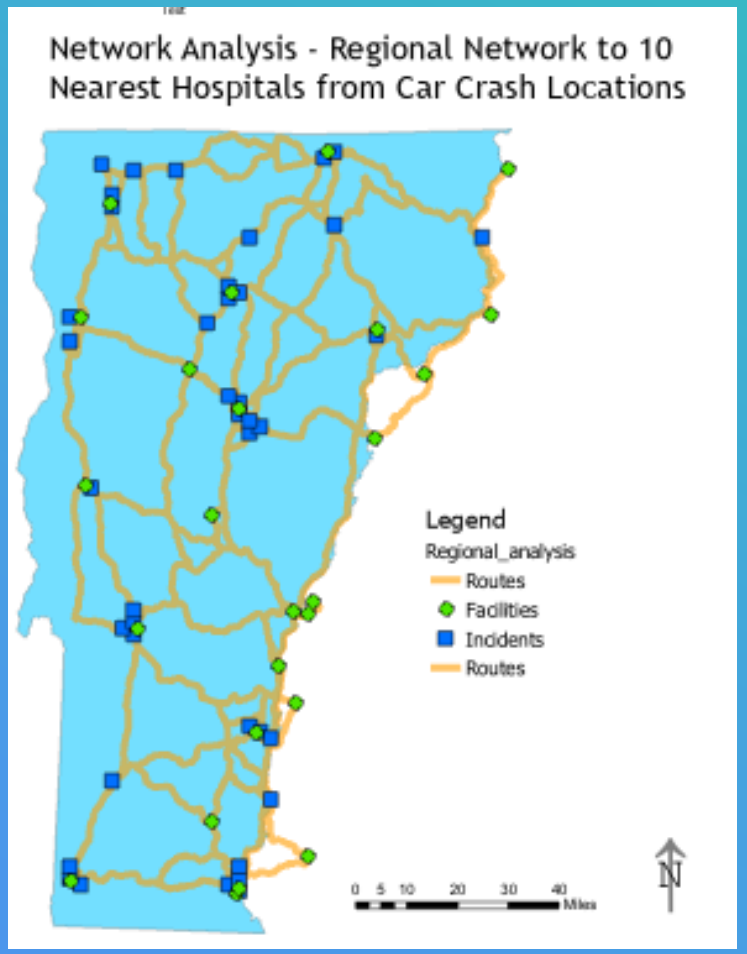
Next, the raster cells are summarized by the inputted fishnet. Using a function created, the centroids are taken from each of the grid cells. Any centroid that has a value less than 0.36 is deleted, as not many car crashes seem to happen in those regions. These will be the "incidents" imputed into network analysis. This network analysis utilizes ArcGIS webservices.

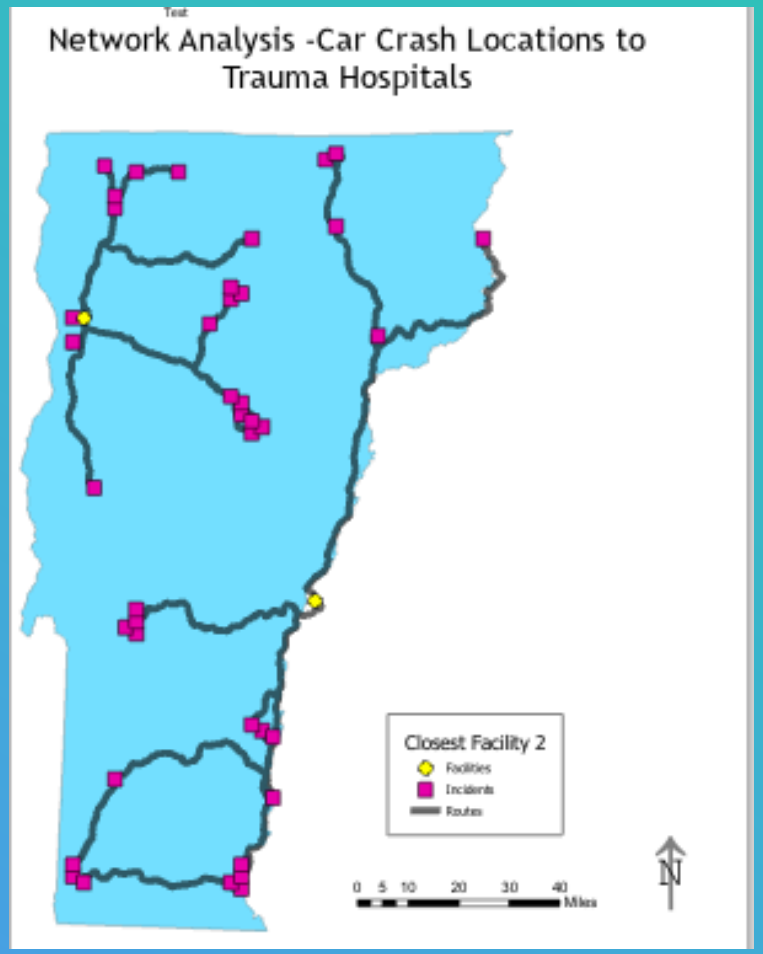
The network analysis has to be inputted by the user. There are three network analysis' performed.

1. Service area, representing 30 minutes from a trauma center. Anything outside of the created polygon is considered underserved.
    1. "incidents" that intersect the data are then removed from the next two network analysis
2. An analysis representing how long it takes for each identified region to get to a trauma center
3. An analysis representing how long it takes for each identified region to get to the 10 nearest hospitals, regardless if they have a trauma center or not.

NETWORK ANALYSIS







Sorting Underserved Car Crash Locations

Total_TravelTime	Shape
125.358571	Point
124.347176	Point
121.851543	Point
119.453523	Point
105.86487	Point
103.439332	Point
98.689864	Point
86.493227	Point
84.856817	Point
80.794779	Point
75.522785	Point
75.04721	Point
73.572625	Point
73.159494	Point
72.694752	Point
68.331285	Point
67.332842	Point
65.915821	Point
62.053537	Point
58.798154	Point
56.894706	Point
56.866942	Point
55.380251	Point
54.662614	Point
54.029441	Point
50.303905	Point
48.528469	Point
48.154399	Point
47.201129	Point

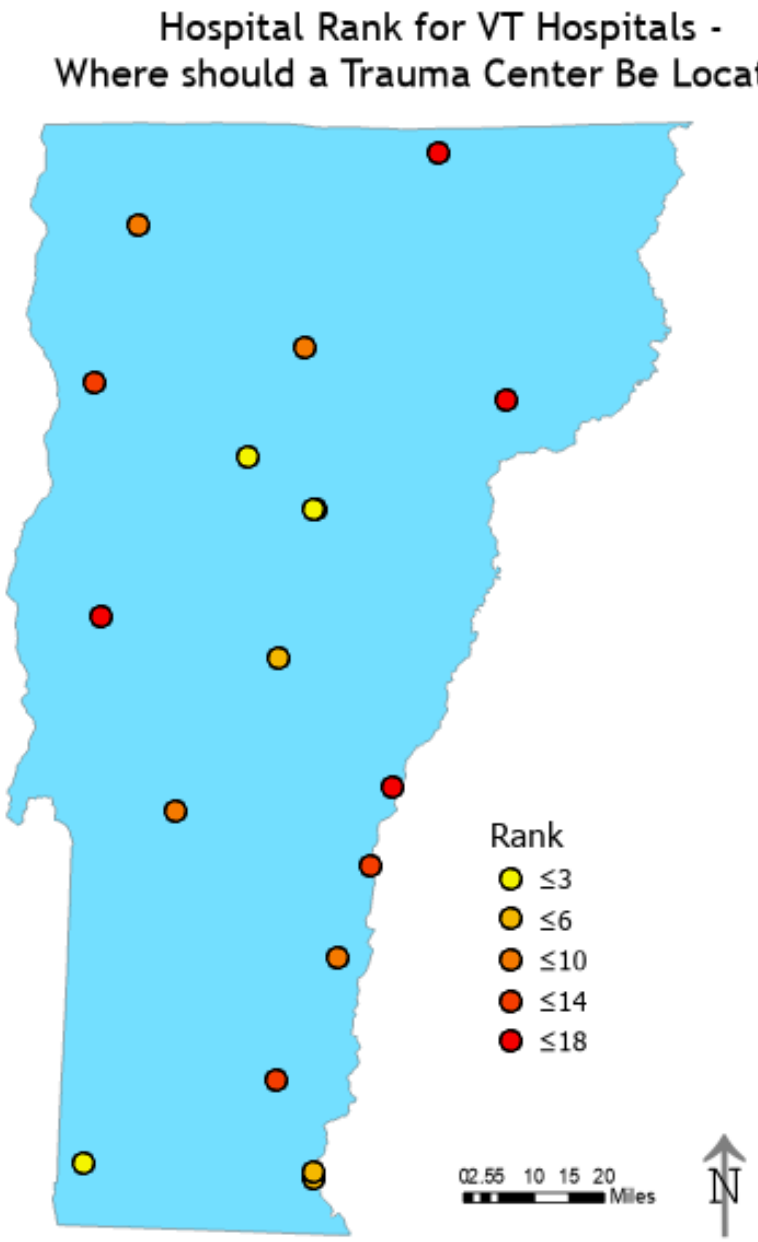
The Network analysis Trauma Center Results (Routes) is joined to the resulting incident feature class, which is then sorted by travel time

Note that there is no associated ranking here as all these areas are underserved. Hence, it does not matter if a location is 1 hr away or 40 min.

Ranking Hospitals without a Trauma Center for Trauma Center Suitability

- Feature class is manipulated to delete hospitals with trauma centers and any hospital that ranks after the trauma center hospital for each incident location
- A field “Count” is created. A value of 1 is added to the field where travel time to a give hospital is less than 35 minutes
- Incident density value from the common car crash locations feature class is joined to table
- Summary Statistic summarizes FacilityID by mean of density value and sum of count
- Values are then joined to regional hospital input file
- Sorted first by count, second by density score
- Hospitals outside of Vermont are deleted
- A Ranking is then assigned based on the sorted order

Sorted Output – Regional Hospitals



Shape	NAME	Rank	Count	Density Score
Point	VERMONT PSYCHIATRIC CARE HOSPITAL	1	9	0.440533
Point	CENTRAL VERMONT MEDICAL CENTER	2	7	0.461511
Point	VERMONT PSYCHIATRIC CARE HOSPITAL	3	7	0.461511
Point	GIFFORD MEDICAL CENTER	4	6	0.462114
Point	BRATTLEBORO MEMORIAL HOSPITAL	5	5	0.475278
Point	BRATTLEBORO RETREAT	6	5	0.475278
Point	COPLEY HOSPITAL	7	5	0.451606
Point	NORTHWESTERN MEDICAL CENTER	8	5	0.433487
Point	SPRINGFIELD HOSPITAL	9	4	0.489883
Point	RUTLAND REGIONAL MEDICAL CENTER	10	4	0.476999
Point	GRACE COTTAGE HOSPITAL	11	4	0.470939
Point	THE UNIVERSITY OF VERMONT MEDICAL CENTER	12	4	0.447044
Point	SOUTHWESTERN VERMONT MEDICAL CENTER	13	3	0.495178
Point	MOUNT ASCUTNEY HOSPITAL & HEALTH CENTER	14	3	0.489883
Point	NORTH COUNTRY HOSPITAL & HEALTH CENTER	15	3	0.450414
Point	NORTHEASTERN VERMONT REGIONAL HOSPITAL	16	2	0.456933
Point	VA MEDICAL CENTER - WHITE RIVER JUNCTION	17	1	0.476083
Point	PORTER MEDICAL CENTER, INC.	18	1	0.466793

The results indicate that a trauma center should be placed in the either Central Vermont or Southwestern Vermont to best prevent car crashes deaths in the state

* Alias changed in table manually

CODE!

```
# im# import packages
import sys
import string
import os
import arcpy
import numpy
import traceback
import math
import csv
from arcpy.sa import *
from arcpy.na import *
arcpy.env.scratchWorkspace = "C:/Users/Kyle McCarthy/Documents/ArcGIS/Projects/traumafinal/traumafinal.gdb"

arcpy.env.overwriteOutput = True

roads = arcpy.GetParameterAsText(0)
car = arcpy.GetParameterAsText(1)
grid = arcpy.GetParameterAsText(2)
table = arcpy.GetParameterAsText(3)
service_area_layer = arcpy.GetParameterAsText(4)
facilities2 = arcpy.GetParameterAsText(5)
network_layer = arcpy.GetParameterAsText(6)
network_layer2= arcpy.GetParameterAsText(7)
facilities = arcpy.GetParameterAsText(8)
Sorted_output = arcpy. GetParameterAsText(9)

# Adding fields to the roads shapefile, which will be manipulated
arcpy.AddField_management(roads, "x1", "LONG")
arcpy.AddField_management(roads, "x2", "LONG")
arcpy.AddField_management(roads, "y1", "LONG")
arcpy.AddField_management(roads, "y2", "LONG")
arcpy.AddField_management(roads, "Angles", "DOUBLE")
```

```
#Getting the Extent min and max for y and x.
arcpy.CalculateGeometryAttributes_management(roads, [["x1", "EXTENT_MIN_X"]])
arcpy.CalculateGeometryAttributes_management(roads, [["x2", "EXTENT_MAX_X"]])
arcpy.CalculateGeometryAttributes_management(roads, [["y1", "EXTENT_MIN_Y"]])
arcpy.CalculateGeometryAttributes_management(roads, [["y2", "EXTENT_MAX_Y"]])

# Calculating Azimuth
arcpy.CalculateField_management(roads, "Angles", '180 + math.atan2((!y1! - !y2!),(!x1! - !x2!)) * (180 / math.pi)')

# Polyline to Raster
arcpy.PolylineToRaster_conversion(roads, 'Angles', "roads", "", "", 400)

# Widening the roads
inRaster = "roads"
neighborhood = NbrRectangle(2, 2, "CELL")

outFocalStat = FocalStatistics(inRaster, neighborhood, "MEDIAN", "DATA")
outFocalStat.save("focal")

# Dividing value by 1000000000
minimized_value = Divide("focal", 1000000000)

# Creating a reclassify list
reclassList = [[10, 5], [11, 5], [12, 4], [13, 3], [14, 2], [16, 1]]
try:
    # Point Density Function
    def pdensity(features, output, val1):
        # Setting point density function parameters
        # Note High cell size -- probably not as accurate, but my computer cannot handle much less
        cellSize = 400
        radius = 700

        #Search radius function
        myNbrCirc = NbrCircle(radius, "MAP")

        # Point density function
        ptd = PointDensity(features, "NONE", cellSize,
                           myNbrCirc, "SQUARE_METERS")
        # alter raster file by creating larger values that are easy to read.
        rasterOUT = Raster(ptd)
        rasterOUT = Abs(Int(Ln(Plus(rasterOUT, 0.0000001))))
        # Remap values into 4 distinct categories
        if len(val1) >= 1:
            outReclass = Reclassify(rasterOUT, "Value",
            RemapValue(val1))
            # Save
            return outReclass.save("density")
        else:
            return ptd.save("density")

    pdensity(car, "density", reclassList)

    # Adding the road raster value to the density raster value
    outCellStats = CellStatistics(["density", minimized_value], "SUM", "DATA")
```

```
    # Shifing cells to follow direction of roads.
    # Calculates mean of surrounding cells for each pixel #that do not have a direction associated with it,
    # or do not trend N-S or E-W take the mean of all values with its neighborhood.
    #The code ensures that outliers are not included in the output,
    #which is when the raster value is below a mean specified standard deviation from neighboring values.

    InputGridName       = outCellStats
    InputArray          = arcpy.RasterToNumPyArray(InputGridName)
    InputArray          = InputArray.astype(float)
    HowManyRows         = InputArray.shape[0]
    HowManyColumns      = InputArray.shape[1]

        # Initialize an OutputArray that is similar to that InutArray but filled with zeroes
    OutputArray         = numpy.zeros_like(InputArray)

        # Get number of standard deviations to be used in defining outliers
    Limit               = 1

        # Initialize vertical and horizontal offests for the nine pixels within each neighborhood
        # Moves pixels N-S for pixels along N-S bearing roads
    RowShift            = [0, 0, 0, 0, 0, 1, -1, -1, -1 ]
    ColumnShift         = [0, 2, 0, -2, 0, 1,  0, -1, 0 ]

    # Moves Pixels E-W for pixels along E-W bearing roads
    RowShift1            = [0, 0, 2, 0,-2, -1, 0, 1, 0 ]
    ColumnShift1         = [0, 0, 0, 0, 0, -1,  -1,  1, 1]

    RowShift2               = [0,-1, 0, 1, 0,-1,-1, 1, 1]
    ColumnShift2             = [0, 0, 1, 0,-1,-1, 1, 1,-1]

        # Loop through rows and columns of pixels
    for ThisRow in range(HowManyRows):
        for ThisColumn in range(HowManyColumns):

            HowManyNeighbors    = 0
                # Loop through the nine pixels in each neighborhood to compute their count and sum
            for NextNeighbor in range(9):
                if (NextNeighbor - int(NextNeighbor))*1000000000 > 0 and  (NextNeighbor - int(NextNeighbor))*1000000000 <= 30:
                    NeighborRow         = ThisRow + RowShift1[NextNeighbor]
                    if NeighborRow      < 0 or NeighborRow      >= HowManyRows:    continue
                    NeighborColumn      = ThisColumn + ColumnShift1[NextNeighbor]
                    if NeighborColumn   < 0 or NeighborColumn   >= HowManyColumns: continue
                elif (NextNeighbor - int(NextNeighbor))*1000000000 >=60:
                    NeighborRow         = ThisRow + RowShift[NextNeighbor]
                    if NeighborRow      < 0 or NeighborRow      >= HowManyRows:    continue
                    NeighborColumn      = ThisColumn + ColumnShift[NextNeighbor]
                    if NeighborColumn   < 0 or NeighborColumn   >= HowManyColumns: continue
                else:
                    NeighborRow         = ThisRow + RowShift2[NextNeighbor]
                    if NeighborRow      < 0 or NeighborRow      >= HowManyRows:    continue
                    NeighborColumn      = ThisColumn + ColumnShift2[NextNeighbor]
                    if NeighborColumn   < 0 or NeighborColumn   >= HowManyColumns: continue


                    # Increment neighborhood sum and neighbor count
                OutputArray[ThisRow][ThisColumn] = OutputArray[ThisRow][ThisColumn] + InputArray[NeighborRow][NeighborColumn]
                HowManyNeighbors = HowManyNeighbors + 1
                # Divide neighborhood sum by neighbor count to get mean of all pixels in neighborhood
            MeanOfAllNeighbors = OutputArray[ThisRow][ThisColumn] / HowManyNeighbors
            OutputArray[ThisRow][ThisColumn] = 0
```

```
    # Loop through the nine pixels in each neighborhood to compute their standard deviation
for NextNeighbor in range(9):
    if (NextNeighbor - int(NextNeighbor))*1000000000 > 0 and  (NextNeighbor - int(NextNeighbor))*1000000000 <= 30:
        NeighborRow         = ThisRow + RowShift1[NextNeighbor]
        if NeighborRow      < 0 or NeighborRow      >= HowManyRows:    continue
        NeighborColumn      = ThisColumn + ColumnShift1[NextNeighbor]
        if NeighborColumn   < 0 or NeighborColumn   >= HowManyColumns: continue
    elif (NextNeighbor - int(NextNeighbor))*1000000000 >= 60:
        NeighborRow         = ThisRow + RowShift[NextNeighbor]
        if NeighborRow      < 0 or NeighborRow      >= HowManyRows:    continue
        NeighborColumn      = ThisColumn + ColumnShift[NextNeighbor]
        if NeighborColumn   < 0 or NeighborColumn   >= HowManyColumns: continue
    else:
        NeighborRow         = ThisRow + RowShift2[NextNeighbor]
        if NeighborRow      < 0 or NeighborRow      >= HowManyRows:    continue
        NeighborColumn      = ThisColumn + ColumnShift2[NextNeighbor]
        if NeighborColumn   < 0 or NeighborColumn   >= HowManyColumns: continue

        # Increment neighborhood sum of squared deviations
    DeviationOfThisNeighbor         = InputArray[NeighborRow][NeighborColumn] - MeanOfAllNeighbors
    SquaredDeviationOfThisNeighbor  = DeviationOfThisNeighbor * DeviationOfThisNeighbor
    OutputArray[ThisRow][ThisColumn] = OutputArray[ThisRow][ThisColumn] + SquaredDeviationOfThisNeighbor
    # Divide neighborhood sum by neighbor count to get mean squared deviation of all pixels in neighborhood
MeanSquaredDeviationOfAllNeighbors  = OutputArray[ThisRow][ThisColumn] / HowManyNeighbors
StandardDeviation                   = math.sqrt(MeanSquaredDeviationOfAllNeighbors)
OutputArray[ThisRow][ThisColumn]    = 0

HowManyNeighbors                    = 0
    # Loop through the non-outlier pixels in each neighborhood to compute their count and sum
for NextNeighbor in range(9):
    if (NextNeighbor - int(NextNeighbor))*1000000000 > 0 and  (NextNeighbor - int(NextNeighbor))*1000000000 <= 30:
        NeighborRow         = ThisRow + RowShift1[NextNeighbor]
        if NeighborRow      < 0 or NeighborRow      >= HowManyRows:    continue
        NeighborColumn      = ThisColumn + ColumnShift1[NextNeighbor]
        if NeighborColumn   < 0 or NeighborColumn   >= HowManyColumns: continue
    elif (NextNeighbor - int(NextNeighbor))*1000000000 >= 60:
        NeighborRow         = ThisRow + RowShift[NextNeighbor]
        if NeighborRow      < 0 or NeighborRow      >= HowManyRows:    continue
        NeighborColumn      = ThisColumn + ColumnShift[NextNeighbor]
        if NeighborColumn   < 0 or NeighborColumn   >= HowManyColumns: continue
    else:
        NeighborRow         = ThisRow + RowShift2[NextNeighbor]
        if NeighborRow      < 0 or NeighborRow      >= HowManyRows:    continue
        NeighborColumn      = ThisColumn + ColumnShift2[NextNeighbor]
        if NeighborColumn   < 0 or NeighborColumn   >= HowManyColumns: continue
    # Increment neighborhood sum and neighbor count
    OutlierLimit = StandardDeviation * Limit
    if InputArray[NeighborRow][NeighborColumn] > (MeanOfAllNeighbors + OutlierLimit): continue
    if InputArray[NeighborRow][NeighborColumn] < (MeanOfAllNeighbors - OutlierLimit): continue
    OutputArray[ThisRow][ThisColumn] = OutputArray[ThisRow][ThisColumn] + InputArray[NeighborRow][NeighborColumn]
    HowManyNeighbors = HowManyNeighbors + 1
    # Divide neighborhood sum by neighbor count to get mean of all pixels in neighborhood
    OutputArray[ThisRow][ThisColumn]  = OutputArray[ThisRow][ThisColumn] / HowManyNeighbors
```

```
    # Create output grid from that new array
InputGrid           = arcpy.Raster(InputGridName)
gridExtent          = InputGrid.extent
lowerleftPoint      = gridExtent.lowerLeft
gridResolution      = InputGrid.meanCellWidth
outputGrid      = arcpy.NumPyArrayToRaster(OutputArray,lowerleftPoint,gridResolution)
arcpy.DefineProjection_management(outputGrid, arcpy.SpatialReference(2852))

# Zonal statistics -- summarizing by grid cell

outZSaT = ZonalStatisticsAsTable(grid, "FID", outputGrid, table, "DATA", "MEAN")

# Finding Centroid of each grid cell in inputted grid file
# Setting serach cursor
cursor = arcpy.da.SearchCursor(grid, "SHAPE@XY")
centroid_coords = []
for feature in cursor:

    centroid_coords.append(feature[0])

    point = arcpy.Point()
    # Creating point geometry list
    pointGeometryList = []

for pt in centroid_coords:
    point.X = pt[0]
    point.Y = pt[1]
    # Appending centroid coordinates
    pointGeometry = arcpy.PointGeometry(point)
    pointGeometryList.append(pointGeometry)

# Copying features
arcpy.CopyFeatures_management(pointGeometryList, "centroidfa.shp")
# Defining projection
arcpy.DefineProjection_management("centroidfa.shp", arcpy.SpatialReference(2852))

# Joining zoning statitstics to centroid
t = arcpy.AddJoin_management("centroidfa.shp", "FID", outZSaT, "FID", "KEEP_ALL")
arcpy.CopyFeatures_management(t, "features.shp")

# Function to add layer to a map
def layertomap(filename):
    nameOfOutputLayer     = (arcpy.Describe(filename).file)[:-4]
    arcpy.MakeFeatureLayer_management(filename, nameOfOutputLayer)

    nameOfOutputLayerFile = filename[:-4] + ".lyrx"
    arcpy.SaveToLayerFile_management(nameOfOutputLayer, nameOfOutputLayerFile, "ABSOLUTE")
    currentProject        = arcpy.mp.ArcGISProject("CURRENT")
    currentMap            = currentProject.activeMap
    Centroid            = arcpy.mp.LayerFile(nameOfOutputLayerFile)
    currentMap.addLayer(Centroid,"TOP")

# Addinc centroid layer to map

layertomap("features.shp")
```

```python
    # Referencing current map
    currentProject  = arcpy.mp.ArcGISProject("CURRENT")
    maps = currentProject.listMaps()[0]
    i = 0
    #Iterating Through layers in Map
    for lyr1 in maps.listLayers():
        # Only select top layer
        if i == 0:
            i += 1
            # Initializing cursor
            with arcpy.da.UpdateCursor(lyr1, ["VT_Highw_4"]) as cursorb:
            #Delete all rows that have value less than 15
                for row in cursorb:
                    if row[0] < 0.36:
                        cursorb.deleteRow()

    # Add trauma locations as facilities to service network layer
    arcpy.na.AddLocations(service_area_layer, "Facilities", facilities2,
        "Name Name #;Attr_Minutes VisitTime 0;CurbApproach # 0")

    #Solving Service Network Layer
    Solve(service_area_layer)

    #Referencing current map again
    currentProject  = arcpy.mp.ArcGISProject("CURRENT")
    maps = currentProject.listMaps()[0]
    i = 0
    # Iterating Through layers in Map
    for lyr2 in maps.listLayers():
        # Only loop through first layer
        if i == 0:
            #Save service layer to layer file
            m = arcpy.SaveToLayerFile_management(service_area_layer, "layer.lyrx")
            #Referencing Layerfile
            p = arcpy.mp.LayerFile(m)
            i += 1
            for lyr in p.listLayers():
                if lyr.name == "Polygons":
                    # Initializing searchCursor in top layer on map
                    with arcpy.da.SearchCursor(lyr2, ["VT_Highw_4"]) as cursor22:
                        for row in cursor22:
                            # Deleting features within the service area boundary
                            arcpy.SelectLayerByLocation_management(lyr2, 'WITHIN',lyr)
                            arcpy.DeleteFeatures_management(lyr2)

            # Adding locations and incidents to Regional Hospital Network Layer
            arcpy.na.AddLocations(network_layer, "Facilities", facilities,
                      "Name Name #;Attr_Minutes VisitTime 0;CurbApproach # 0")
            arcpy.na.AddLocations(network_layer, "Incidents", lyr2)
            # Solving Regional Hospital Network Layer
            Solve(network_layer, "SKIP", "Continue")


            # Adding Facilities and incidents to Trauma Center Network Layer

            arcpy.na.AddLocations(network_layer2, "Facilities", facilities2,
                      "Name Name #;Attr_Minutes VisitTime 0;CurbApproach # 0")
            arcpy.na.AddLocations(network_layer2, "Incidents", lyr2,
                      "Name Name #;Attr_Minutes VisitTime 0;CurbApproach # 0")
            #Solving Trauma Center Network Layer
            Solve(network_layer2, "SKIP", "Continue")
```

```python
    # Referencing top layer in current map
    currentProject        = arcpy.mp.ArcGISProject("CURRENT")
    maps = currentProject.listMaps()[0]
    i = 0
    for lyr1 in maps.listLayers():
        # only list first layer
        if i == 0:
            # Referencing Network Layer - Regional
            networkm = arcpy.SaveToLayerFile_management(network_layer2, "help.lyrx")
            networkp = arcpy.mp.LayerFile(networkm)
            i += 1
            # Creating IncidentId to join to
            arcpy.AddField_management(lyr1, "IncidentID", "LONG")
            # Calculating Incident ID
            arcpy.CalculateField_management(lyr1, "IncidentID", '!FID! + 1')
            # JOining Trauma Center Route Results to Incident Shapefile
            for lyr2 in networkp.listLayers():
                if lyr2.name == "Routes":
                    arcpy.JoinField_management(lyr1, "IncidentID", lyr2, "IncidentID")

    # Referencing Regional- Hospital Closest Facility layer
    networkm = arcpy.SaveToLayerFile_management(network_layer_regional, "helme.lyrx")
    networkp = arcpy.mp.LayerFile(networkm)
    # Referencing Routes layer within service layer
    with arcpy.da.UpdateCursor(networkp.listLayers()[4], ["FacilityID", "IncidentID", "FacilityRank"]) as cursor22:
        i = 105
        Ranking = 0
        for row in cursor22:
            # if i equals the incident id
            if row[1] == i:
                # Incident ID cannot be equalt to Dartmoth's Hospital or UVM's
                if row[0] == 7 or row[0] == 16:
                    #If it is set ranking to the ranking value of Darmouth or UVM
                    Ranking = row[2]
                if Ranking >= row[2]:
                    # If ranking is greater or equal to Ranking, delete Row
                    cursor22.deleteRow()
            # If i no longer equals incident ID, add 1 to i and set Ranking back to 0
            if row[1] > i:
                i += 1
                Ranking = 0

    # Referencing Current Map
    currentProject  = arcpy.mp.ArcGISProject("CURRENT")
    maps = currentProject.listMaps()[0]
    i = 0
    # Iterating Through layers in Map
    for lyr2 in maps.listLayers():
        i += 1
        # Only loop through first layer
        if i == 1:
            # Add join field to layer
            arcpy.AddField_management(lyr2, "Join_field", "LONG")
            # Calculate join field to be the same as the incident ID in the network layer
            arcpy.CalculateField_management(lyr2, "Join_field", "!FID! + 1")
            for lyr in networkp.listLayers():
                if lyr.name == "Routes":
                    # Join incidentID information to network layer
                    arcpy.JoinField_management(lyr, "IncidentID",
                                               lyr2, "Join_field", ["VT_Highw_4"])
```

```
    # SEtting parameters for summary statistics
    in_table = networkp.listLayers()[4]
    out_table = "stats1.dbf"
    case_field = "FacilityID"

    #Summary Statistics
    arcpy.Statistics_analysis(in_table, out_table,  [["Counts", "SUM"], ["VT_Highw_4", "MEAN"]], case_field)
    # Add Join field in facilities
    arcpy.AddField_management(facilities, "Join_", "LONG")
    # Join Summary Statiscs to Facilities
    arcpy.CalculateField_management(facilities, "Join_", "!FID! + 1")
    arcpy.JoinField_management(facilities, "Join_", "stats1.dbf", "FacilityID", ["SUM_Counts", "MEAN_VT_Hi"])

    # Sort Fields
    arcpy.Sort_management(facilities, Sorted_output, [["SUM_Counts", "DESCENDING"], ["MEAN_VT_Hi", "DESCENDING"]])
    # Add sorted fields to map
    layertomap("Ranking_Sort1.shp")

    # Reference current project
    currentProject        = arcpy.mp.ArcGISProject("CURRENT")
    maps = currentProject.listMaps()[0]
    i = 0
    #Iterate through layers
    for lyr1 in maps.listLayers():
        # Stop at top layer
        if i == 0:
                # Delete any hospital outside of VT
                with arcpy.da.UpdateCursor(lyr1, ["State"]) as cursory:
                    for row in cursory:
                        if row[0] == "NH":
                                cursory.deleteRow()

    # Add Ranking field
    arcpy.AddField_management(Sorted_output, "Ranking", "LONG")
    # Calculate Ranking
    arcpy.CalculateField_management(Sorted_output, "Ranking", "!FID! + 1")

    # Sort Incidents
    arcpy.Sort_management("features", Sorted_output2, ["Total_TravelTime", "DESCENDING"])



except Exception as e:
    # If unsuccessful, end gracefully by indicating why
    arcpy.AddError('\n' + "Script failed because: \t\t" + e.args[0] )
    # ... and where
    exceptionreport = sys.exc_info()[2]
    fullermessage   = traceback.format_tb(exceptionreport)[0]
    arcpy.AddError("at this location: \n\n" + fullermessage + "\n")
```

Sources

1. Arcgis.com : https://www.arcgis.com/index.htm

2. Stack Exchange:Hot Questions - Stack Exchange

3. CDC Motor Vehicle Crash Deaths | VitalSigns | CDC