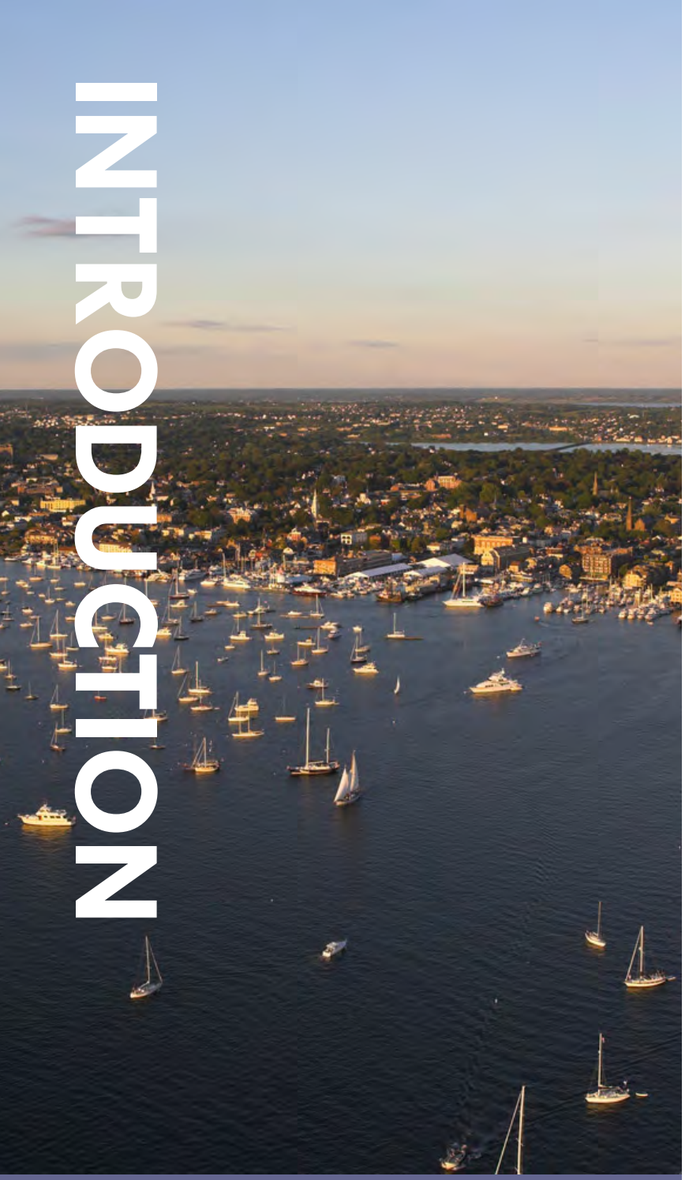
SOCIAL VULNERABILITY TO SEA LEVEL RISE AND STORM SURGES

-- Taking Aquidneck Island as An Example

Xintian Stella Li
Geospatial Software Design - CPLN 670
Instructor: Prof. Dana Tomlin
December 17th, 2020

TABLE OF CONTENTS



1. INTRODUCTION

1.1 OVERVIEW OF AQUIDNECK ISLAND

1.2 CLIMATE CHANGE AND ENVIRONMENTAL THREATS

1.3 GOALS & PURPOSES

1.4 UNIT OF ANALYSIS

Aquidneck Wave

1.1 Overview of Aquidneck Island

Welcome to Aquidneck Island! At 38 square miles, Aquidneck Island is the largest island in the Narragansett Bay and the subject of this study. It is home to 3 municipalities: Newport in the south, Portsmouth in the north and Middletown in the middle. Today, the island is largely known as a picturesque, coastal retreat that has attracted visitors to its beautiful beaches for centuries.

1.2 Climate Change Posed Great treats

Many of the island's residents and physical assets are left exposed to sea level rise and increasingly common and destructive storm events. This poses a unique threat to Aquidneck Island's livelihood. As measured at the Newport tide gauge, the sea level has risen over 10 inches since 1930. Aquidneck residents are already seeing repercussions of these changes in increasingly common flooding and coastal erosion. According to the National Oceanic and Atmospheric Administration (NOAA) 2017 projections based on the "high curve", the sea level rise projections for Rhode Island is 3.25 feet by 2050. Aquidneck Island is under great risk of flooding and inundation.

1.3 Goals & Purposes

In current planning practice, when modeling the vulnerability to Sea Level Rise (SRL), with most efforts focused on examining and reducing biophysical vulnerability and the vulnerability of the built environment, the social-economy conditions of the local communities are seldomly included. The ignorance of the local socia vulnerability will lead to the failure of climate-change reactions. Because although different groups of a society may share a similar exposure to a natural hazard, the hazard has varying consequences for these groups, since they have diverging capacities and abilities to handle the impact of a hazard. Therefore, more attention should be paid to include social vulnerability in the disaster risk evaluation process.

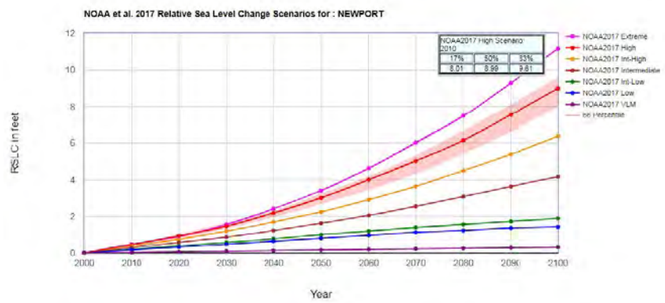
In this term project, the geographic, socio-economic and built environment related characteristics of the Aquidneck Island will be included to analyze the overall risks of flooding hazards. The goal of this study is to utilize this quantitative model to guide managed retreat from the most vulnerable communities in advance and to better allocate resources during and after hazards. This study will help to better understand how various phenomena (hydrological, meteorological, geophysical, social, political and economic) affect daily lives.

1.4 Unit of Analysis

A 200-meter grid map of Aquidneck Island will be created for the study use. With such a small scale analysis unit, the local environmental and demographic data could be depicted and more detaild recommendations could be generated.

A grid map of Aquidneck Island

Rhode Island and Aquidneck Island Context



NOAA 2017 Sea Level Rise Change Scenarios for Newport

METHODS & PROJECT

2. METHODS & PROJECT OUTLINE

2.1 DECISION MAKING PROCESS

2.2 SOCIAL VULNERABILITY MODELING

2.3 TOOLS & DATA

A grid map of Aquidneck Island

2.1 Decision Making Process

Because there are over 3600 residents live in areas exposed to storm-related inundation and this number will go up to 12,000 by 2100, we need a mechanism to set priorities that need immediate attention. To organize community actions and spendings, the writer proposes a decision making process that integrates spatial and qualitative analysis.

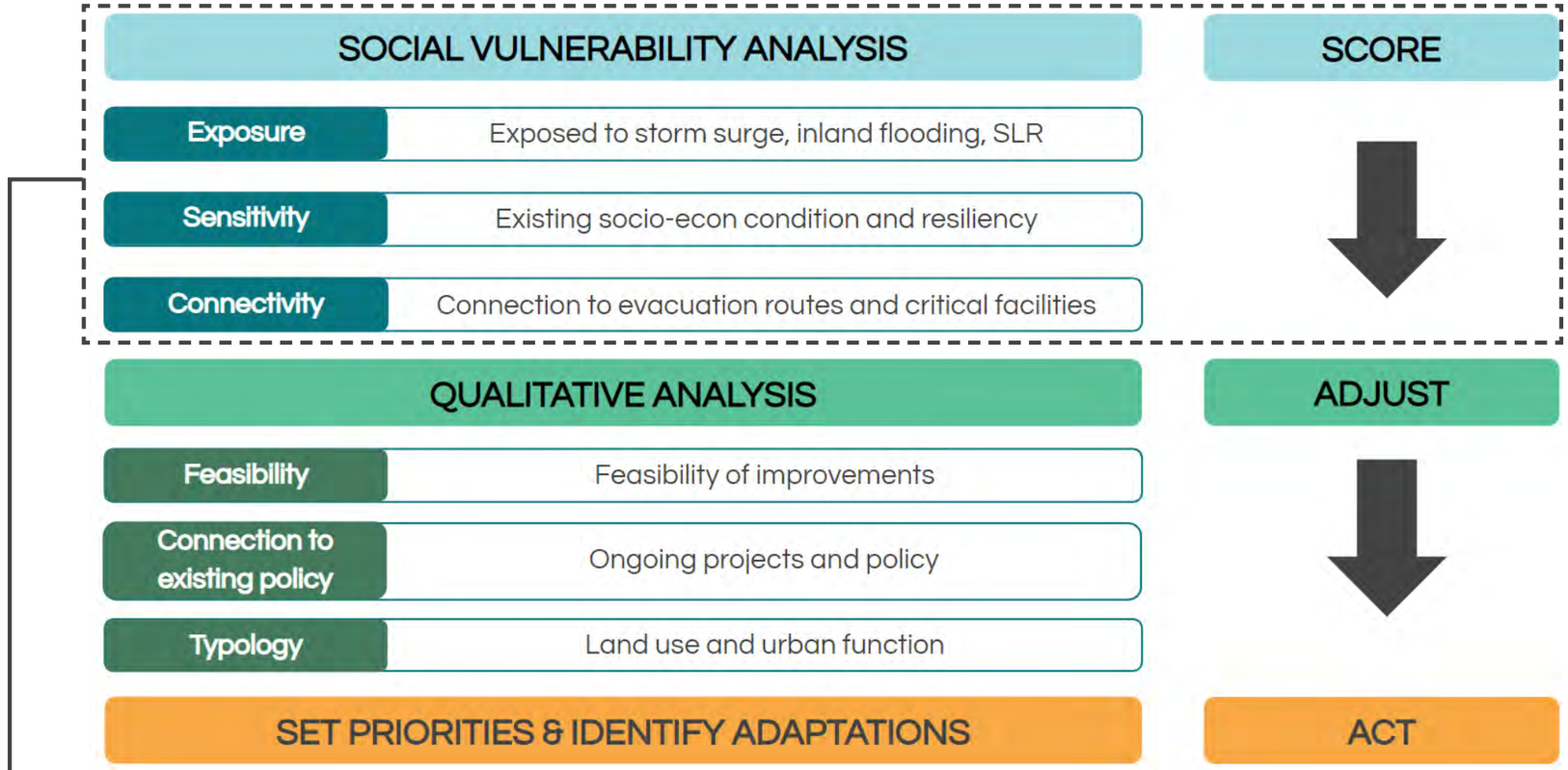
The flow chart shows the decision making process. By following this process, the prioritized communities which should see interventions first will be decided. First, the local planners should conduct a spatial analysis model to score social vulnerability from three perspectives. Exposure, measures the area's exposure to SLR-caused permanent inundation, inland flooding, and storm surges. Sensitivity, measures the communities' socio-economic resiliency in face of natural hazards. Connectivity, evaluates the area's connection to the whole urban system. This analysis model will help decision makers focus on more vulnerable areas.

Next, local decision-makers should adjust the scores generated from the social vulnerability modeling using qualitative information. I propose three different categories to assess: feasibility, connection of existing policy, and typology. Adjustment using qualitative data can bring back a human touch into the analysis and respond better to issues like funding availability.

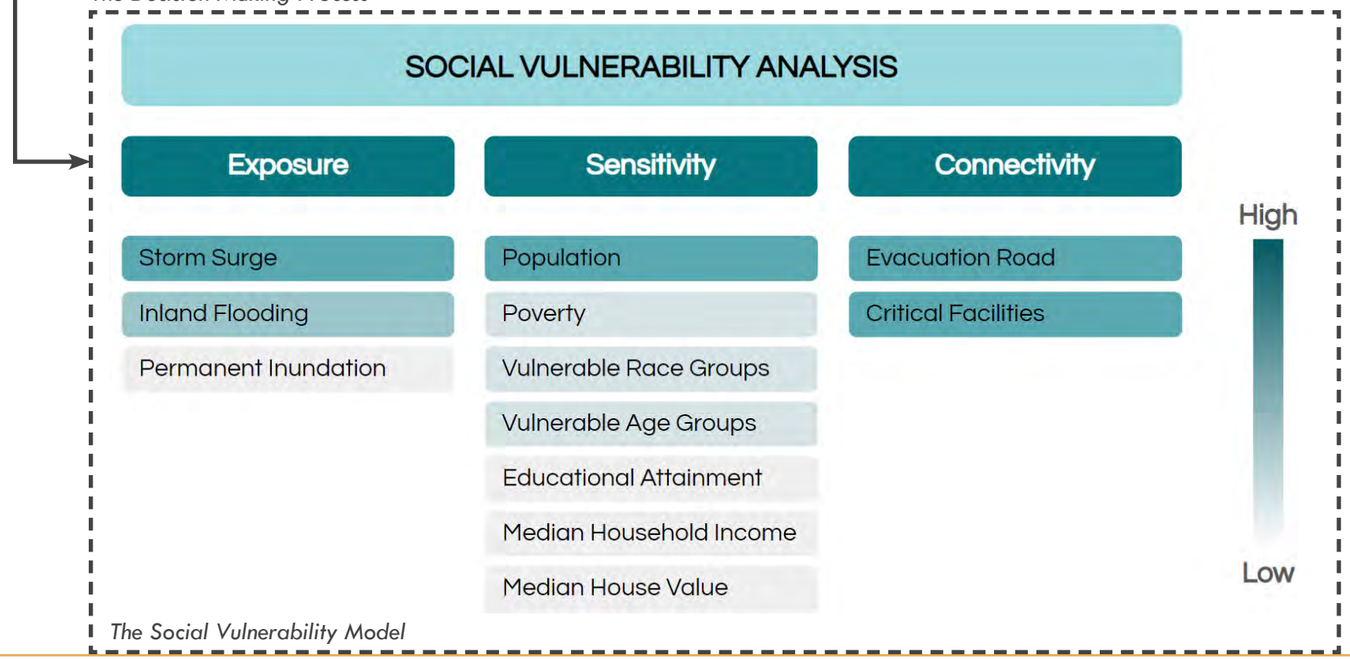
Based on the spatial analysis and qualitative analysis, the decision makers could then come up with the priorities and adaptations. It must be remembered that in these changing conditions, not all assets can be given equal protection.

2.2 Social Vulnerability Modeling

This study will mainly focus on the social vulnerability analysis model. In this model, A set of indicators and index will be developed to score the climate risks on a fine-granular unit of analysis (a 200-meter fishnet). In the following report, the unit of analysis will be referred as "cell". Three indexes will be given different weights and a final social vulnerability score will then be generated.



The Decision Making Process



The Social Vulnerability Model

Exposure = 0.5 * Mean storm surge inundation + 0.2 * Proximity to ponds + 0.2 * Proximity to rivers + 0.1 * Elavation

Exposure Equation

Under the exposure index, three main hazards vulnerability will be estimated. The storm surge score will be generated by calculating the mean inundation depth of each cell during a 100-year storm surge. The deeper the inundation depth, the higher the storm surge score will be. Inland flooding vulnerability is estimated by the proximity to rivers and ponds. If a cell is in close proximity to rivers (10 meters) and ponds (35 meters), it is more prone to inland rivering. Similarly, the future permanent inundation probability is measured by the elevation.

Sensitivity = 0.52 * Population + 0.15 * # of people living under poverty + 0.15 * Racial minorities + 0.15 * # of people < 5 or > 65 + 0.01 * Median household income + 0.01 * Median home value + 0.01 * # of people with a bachelor's degree or above

Sensitivity Equation

When estimating the sensitivity to natural hazards, socio-economic indicators such as population, number of people living under the poverty line, racial minorities, number of people younger than 5 or older than 65, median household income, median house value and the number of people with a bachelor's degree or above will be included. It should be noted that because all the socio-economic data are collecting on the block group level, the Daysymatric Interpolation Method is used to allocate these block group level data proportionally to each cell based on the population in the cell. The cell with worse socio-economic conditions will earn higher sensitivity score.

Connectivity = 0.5 * Critical facilities + 0.5 * Evacuation routes

Connectivity Equation

The connectivity is estimated using two indicators, the critical facilities and the evacuation routes. The critical facilities such as police station, fire station, city hall, emergency shelter are essential to maintain the well-functioning of the city and also support public welfare under urgent conditions. Therefore, the cells that contain critical facilities or evacuation routes will be given a high connectivity score.

Social Vulnerability Score = 0.4 * Exposure + 0.4 * Sensitivity + 0.2 * Connectivity

Social Vulnerability Score Equation

The final social vulnerability score is computed by summing up weighted exposure score, sensitivity score and connectivity score. In this study, the writer assume that exposure and sensitivity are more important than connectivity, hence they are given larger weights.

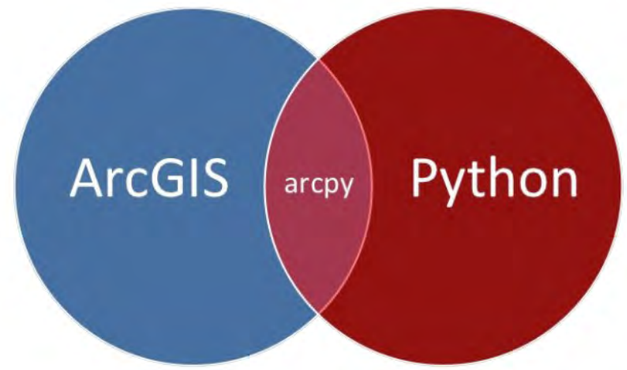
2.3 Tools & Data

ArcPy is a Python site package that provides a useful and productive way to perform geographic data analysis, data conversion, data management, and map automation with Python.

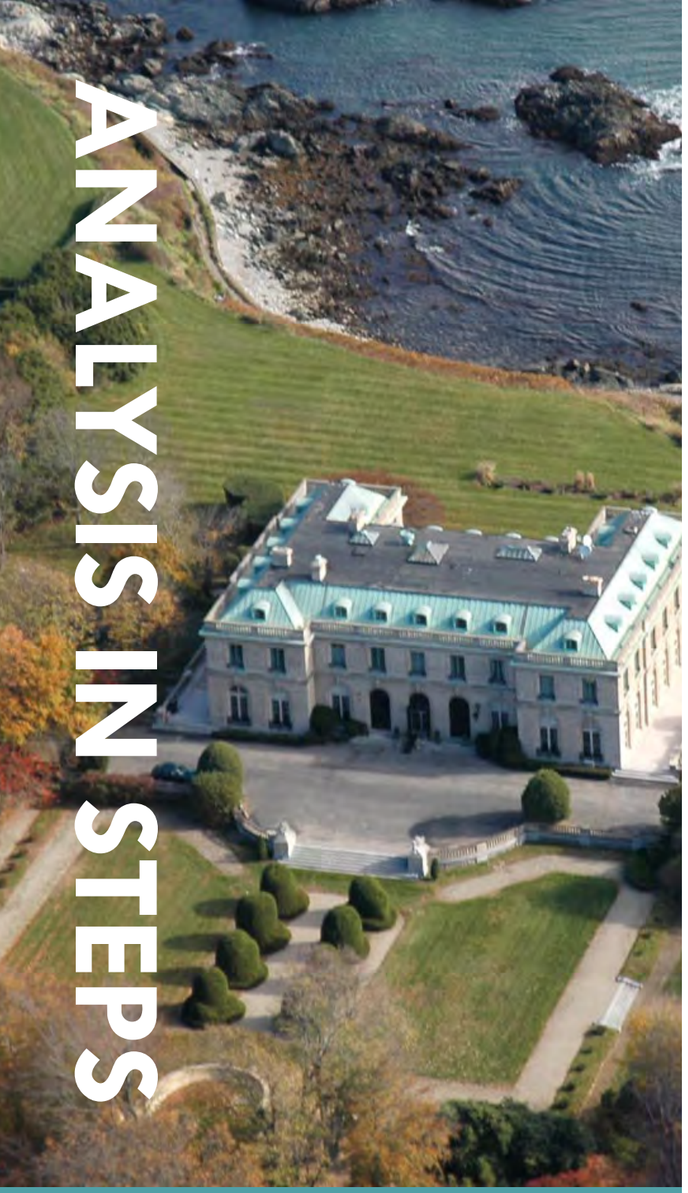
Python is a general-purpose programming language. It is interpreted and dynamically typed and is suited for interactive work and quick prototyping of one-off programs known as scripts while being powerful enough to write large applications in. Using ArcPython to write ArcGIS applications and tools benefit from both modules in ArcGIS and Python. In this study, a series of self-defined and automated spatial analysis tools will be generated using ArcPython script.

Data used in this study involves environmental data, socio economic data and data that depicting built environment. The list of data is as following, more information about data sources will be provided in the Appendix.

- Storm Surge Inundation Depth Raster from STORMTOOLS
- DEM (elevation) data generated from 10 ft Contour Line from RIGIS
- Rivers & Ponds Feature Classes from RIGIS
- Block Group Census from ACS 2018, Census Bureau
- Critical Facilities from RIGIS
- Evacuation Routes from the Community Plan document of three townships (Portsmouth, Middletown and Newport)
- 100 meter Population Raster from Google Earth Engine



ArcPython



ANALYSIS IN STEPS

3. ANALYSIS IN STEPS

3.1 SENSITIVITY

3.2 EXPOSURE

3.3 CONNECTIVITY

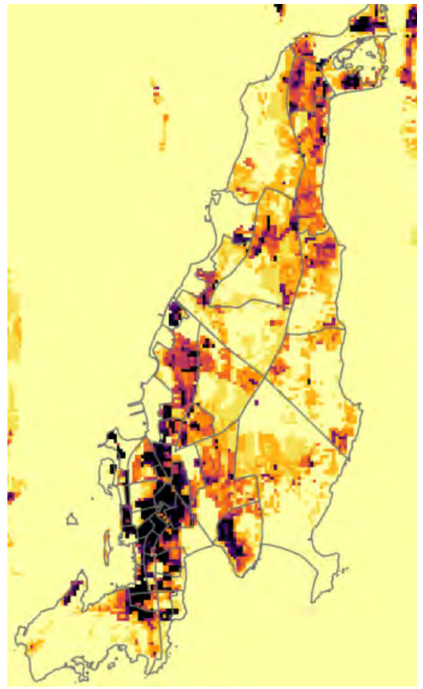
3.4 WEIGHTED SUMMARY

An elevation grid of Aquidneck Island

3.1 Sensitivity

A grid map of Aquidneck Island

Create a fishnet map



100-meter Population Raster

```
## STEP 1: Execute GridIndexFeatures
arcpy.GridIndexFeatures_cartography("Grid200m",blockGroup, "", "", "",
                                    "200 meters", "200 meters")

## STEP 2: Calculate zonal sum population of each cell
arcpy.sa.ZonalStatisticsAsTable("Grid200m","PageNumber",population,"ZonalStable","DAT
```

OID	PageNumber	COUNT	AREA	SUM
1464	1465	4	40000	349
2590	2591	4	40000	343
1645	1646	4	40000	289
1775	1776	4	40000	271
2130	2131	4	40000	255
2131	2132	4	40000	233
1816	1817	4	40000	212
2272	2273	4	40000	210
2204	2205	4	40000	198
1853	1854	4	40000	191
1774	1775	4	40000	189
2203	2204	4	40000	181

Calclate the sum of popualtion in cell

FID	Shape *	Join_Count	Join_Cou	PageNum	grdPOP
1770	Polygon	0	0	1776	271
1989	Polygon	0	0	1817	212
1661	Polygon	0	0	1646	289
1769	Polygon	1	0	1775	189
2221	Polygon	0	0	2131	255
2267	Polygon	0	0	2204	181
1817	Polygon	0	0	1854	191
2268	Polygon	0	9	2205	198
2222	Polygon	0	8	2132	233
1829	Polygon	0	0	1897	118
1335	Polygon	0	0	1606	135
2220	Polygon	0	0	2130	97
2248	Polygon	0	1	2168	122
1810	Polygon	0	0	1812	109
2293	Polygon	1	8	2273	210
2194	Polygon	0	0	2094	97
2193	Polygon	0	0	2093	108
2024	Polygon	0	0	1861	95
2247	Polygon	0	0	2167	82

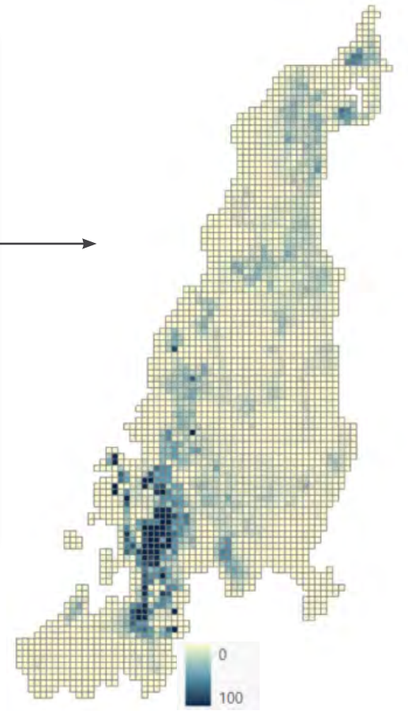
Join the table to the grid

```
tempList = arcpy.ListFields(cleanBlockGroup)
#### 7-2: Loop through all the fields
for field in tempList:
    # do not change those required fields
    if field.required == True:
        continue
    # Split field name at _ symbol
    fieldNames = field.name.split("_")
    # Delete the sufix
    del(fieldNames[0:1])
    # Set the new field value equal to the old one
    expression = "!"+field.name+"!"
    if len(fieldNames) == 1:
        if fieldNames[0] == "SUM":
            # add a new field using the same properties as the original field
            arcpy.AddField_management(cleanBlockGroup, "POP", field.type)
            # calculate the values of the new field
            # set it to be equal to the old field
            arcpy.CalculateField_management(cleanBlockGroup, "POP", expression)
            # delete the old fields
            arcpy.DeleteField_management(cleanBlockGroup, field.name)
            continue
        if field.type == "Integer":
            arcpy.AddField_management(cleanBlockGroup, fieldNames[0], "LONG",20)
            arcpy.CalculateField_management(cleanBlockGroup, fieldNames[0], expression)
            arcpy.DeleteField_management(cleanBlockGroup, field.name)
        elif field.type == "String":
            arcpy.AddField_management(cleanBlockGroup, fieldNames[0], "TEXT",30)
            arcpy.CalculateField_management(cleanBlockGroup, fieldNames[0], expression)
```

Rename the fields of the joined data set

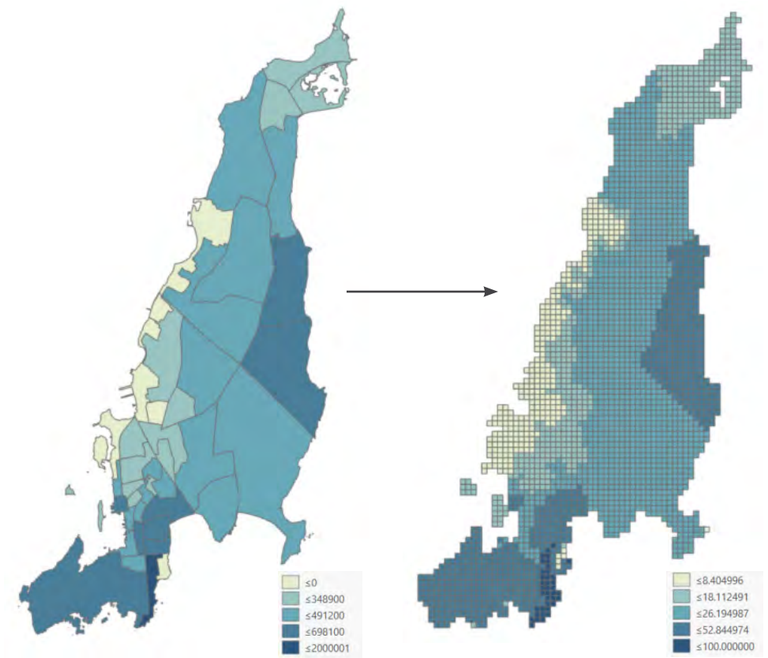
```
## STEP 13: Normalize the fields and scale to 1-100
    # Min-Max Normalization formula:
    # NewValue = (OldValue - min) / (max-min) * 100
nameOfFieldList = ["M_MedHHInc","M_MedHomVal","Old_Yng",
        "highEdu","minRace","thePoor"]
New_nameOfFieldList = ["N_MedHHInc","N_MedHomVal","N_OldYng",
        "N_highEdu","N_minRace","N_thePoor"]
for newName in New_nameOfFieldList:
    arcpy.AddField_management(GridWithCEN,newName,"DOUBLE",5,8)
for nameOfValueField in nameOfFieldList:
    # Convert the attribute table to Numpy Array
    valueFieldArray = arcpy.da.TableToNumPyArray(GridWithCEN,
                                                 nameOfValueField)
    minValue = numpy.min(valueFieldArray[nameOfValueField])
    maxValue = numpy.max(valueFieldArray[nameOfValueField])
    enumerationOfRecords1 = arcpy.UpdateCursor(GridWithCEN)
    for nextRecord in enumerationOfRecords1:
        OldValue = nextRecord.getValue(nameOfValueField)
        normalizedValue = (OldValue - minValue) / (maxValue - mi
        nextRecord.setValue(New_nameOfFieldList[nameOfFieldList.
                            normalizedValue)
        enumerationOfRecords1.updateRow(nextRecord)
    arcpy.AddMessage("\n")
    del nextRecord
    del enumerationOfRecords1
```

Normalize the value of each cell to 1-100

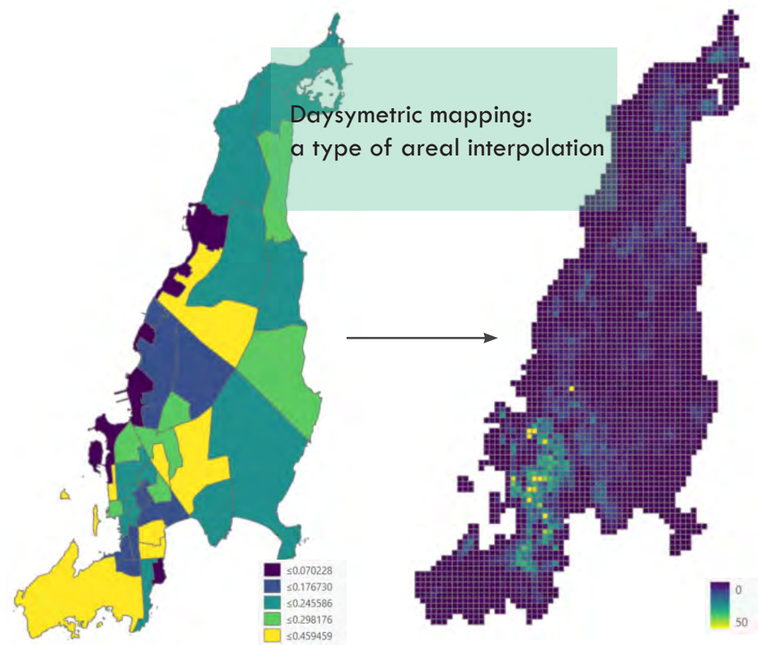


Population Fishnet

3.1 Sensitivity



Median Home Value on Block Groups *Median Home Value on the Grid*



Percent of Old and Young People on Block groups *Percent of Old and Young People on the Grid*

Daysymetric mapping:
a type of areal interpolation

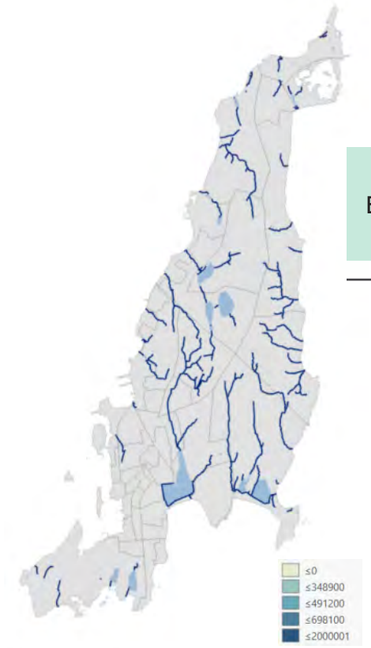
```
## STEP 10: Spatial join the demographic attributes (in the block group shapefile) to the Grid layer
    # Want to join block group to the grid and calculate the mean census data
    # for each cell
    # Output will be the target features, states, with a mean city population field (mcp)
#### 10-1: Create a new fieldmappings and add the two input feature classes.
fieldmappings = arcpy.FieldMappings()
fieldmappings.addTable(cleanGridLayer)
fieldmappings.addTable(cleanBlockGroup)
#### 10-2: Create a list of the fields that we want to join to the Grid layer
spatialJoinFieldList = ["totPop","MedHHInc","MedHomVal","Pct_Oldand","Pct_MinRac","Pct_Bachel","Pct_PoverB","POP"]
#### 10-3: Renew the fieldmap for each field that we want to join to the Grid layer
    # First get the fieldmap from fieldmapping. Each is a field in the cleanBlockGroup feature class.
    # The output will have the cells with the attributes of the block groups.
    # Setting the field's merge rule to mean will aggregate the values for all of the block groups that
    # each cell intersect with into an average value.
    # The field is also renamed to be more appropriate
    # for the output.
for eachField in spatialJoinFieldList:
    # Get each field
    FieldIndex = fieldmappings.findFieldMapIndex(eachField)
    fieldmap = fieldmappings.getFieldMap(FieldIndex)
    # Get the output field's properties as a field object
    field = fieldmap.outputField
    # Rename the field and pass the updated field object back into the field map
    field.name = "M_" + eachField
    field.aliasName = "M_" + eachField
    fieldmap.outputField = field
    # Set the merge rule to mean and then replace the old fieldmap in the mappings object
    # with the updated one
    fieldmap.mergeRule = "mean"
    fieldmappings.replaceFieldMap(FieldIndex, fieldmap)
#### 10-4: Delete fields that are no longer applicable, such as OBJECTID and GEOID of the block groups
# fieldmappings.removeFieldMap(fieldmappings.findFieldMapIndex("cleanBlockGroup_OBJECTID"))
fieldmappings.removeFieldMap(fieldmappings.findFieldMapIndex("GEOID"))
#### 10-5: Run the Spatial Join tool
arcpy.SpatialJoin_analysis(cleanGridLayer,cleanBlockGroup,"GridWithCEN","JOIN_ONE_TO_ONE","KEEP_ALL",fieldmappings,"INTERSECT")
```

Spatial Join the block group level data to the grid.

```
## STEP 12: Create new fields and populate the values for the new field
#### 12-1: Create new fields
# arcpy.AddField_management(GridWithCEN,"Pct_POP","DOUBLE", 20, 8)
arcpy.AddField_management(GridWithCEN,"Old_Yng","DOUBLE", 20, 8)
arcpy.AddField_management(GridWithCEN,"highEdu","DOUBLE", 20, 8)
arcpy.AddField_management(GridWithCEN,"minRace","DOUBLE", 20, 8)
arcpy.AddField_management(GridWithCEN,"thePoor","DOUBLE", 20, 8)
#### 12-2: Create new fields
enumerationOfRecords = arcpy.UpdateCursor(GridWithCEN)
for nextRecord in enumerationOfRecords:
    gridPopulation = nextRecord.getValue("grdPOP")
    blockGroupPopulation = nextRecord.getValue("M_POP")
    Pct_OldandYoung = nextRecord.getValue("M_Pct_Oldand")
    Pct_MinorRace = nextRecord.getValue("M_Pct_MinRac")
    Pct_BachelorAndAbove = nextRecord.getValue("M_Pct_Bachel")
    Pct_BelowPovertyLine = nextRecord.getValue("M_Pct_PoverB")
    Old_Yng = gridPopulation * Pct_OldandYoung
    highEdu = gridPopulation * Pct_BachelorAndAbove
    minRace = gridPopulation * Pct_MinorRace
    thePoor = gridPopulation * Pct_BelowPovertyLine
    nextRecord.setValue("Old_Yng",Old_Yng)
    nextRecord.setValue("highEdu",highEdu)
    nextRecord.setValue("minRace",minRace)
    nextRecord.setValue("thePoor",thePoor)
    enumerationOfRecords.updateRow(nextRecord)
# Add a blank line at the bottom
arcpy.AddMessage('\n')
# Update the records
del nextRecord
del enumerationOfRecords
```

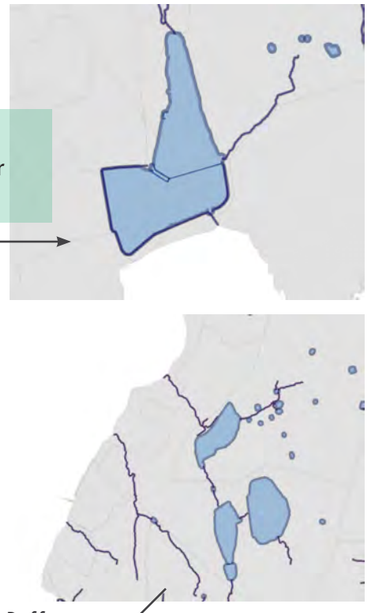
Create new fields in the grid shapefile and populate the fields using daysy-metric map.

3.2 Exposure



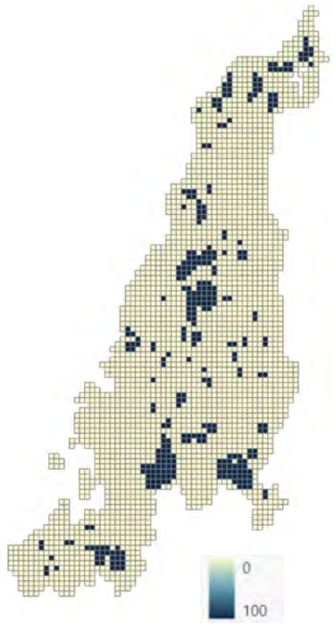
Rivers and Ponds

Buffer



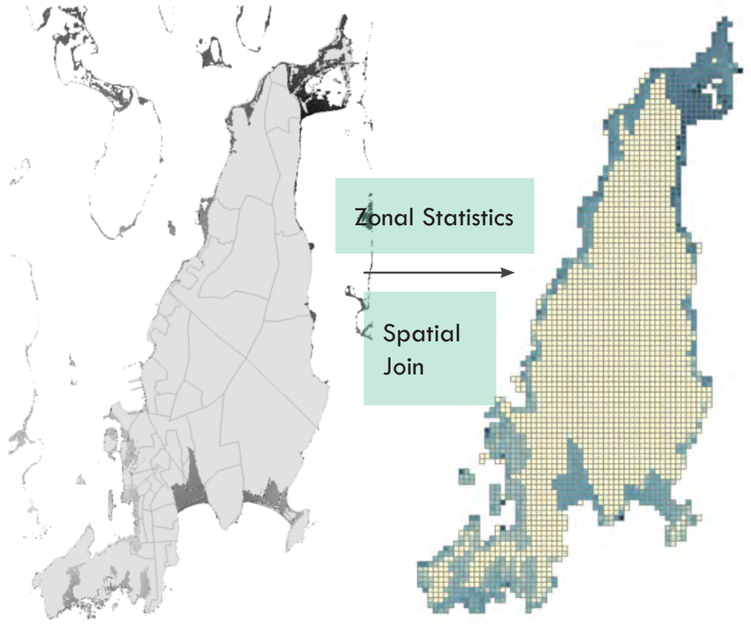
Buffers

Spatial Join



Proximity to Ponds

Proximity to Rivers



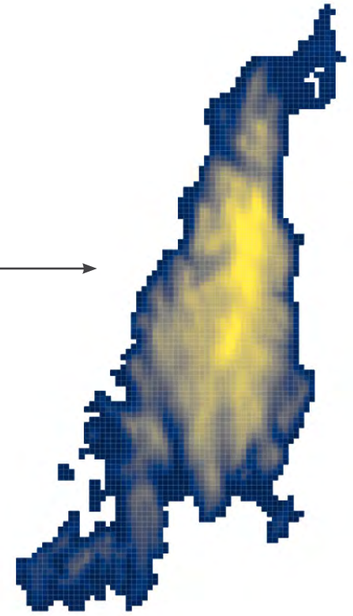
Zonal Statistics

Spatial Join

100 year Storm inundation

Storm Surge Inundation on the Grid

Elevation



Elevation grid of Aquidneck Island

Buffer

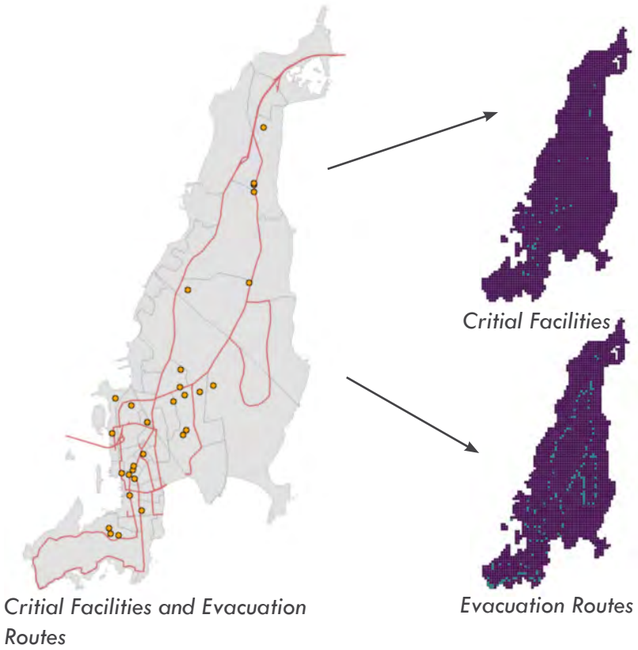
```
## STEP 7: Buffer areas of impact around lakes and stream
arcpy.Buffer_analysis(lakesPonds, "bufferLakesPonds", "15
                      "ALL")
arcpy.Buffer_analysis(riverStreams, "bufferRiverStreams",
                      "ALL")


## STEP 8: Spatial join the buffers to the grid layer
joinedGridWithCEN3 = arcpy.SpatialJoin_analysis(joinedGri
                          "joinedGridWithCEN3","JOIN_ONE_TO
joinedGridWithCEN4 = arcpy.SpatialJoin_analysis(joinedGri
                          "joinedGridWithCEN4","JOIN_ONE_TO
```

Spatial Join

Rename, Normalize, and Add to the attribute table

ifLakesPon	ifStrmsRiv	N_grdPOP	N_eleva	N_inundat	Shape_Le_1
0	0	77.65043	8.087559	0	800
0	0	60.744986	2.753067	0	800
0	0	82.808023	9.5252	0	800
0	0	54.154728	39.4645	0	800
0	0	73.065903	3.828955	0	800
0	0	51.862464	4.288818	0	800
0	0	54.727794	1.824379	77.691625	800
0	0	56.733524	5.681715	0	800
0	0	66.762178	9.162225	0	800
0	1	33.810888	2.045722	74.132439	800
0	0	38.681948	8.96777	58.405372	800
0	0	27.793696	8.693264	0	800
0	0	34.95702	9.640081	0	800
0	0	31.232092	1.865796	76.548537	800
0	0	60.17192	2.057739	0	800
0	0	33.524355	4.602636	0	800
0	0	30.945559	4.259283	0	800
0	0	27.22063	38.78927	0	800
0	0	23.495702	18.4726	0	800
0	0	42.979943	3.443914	0	800
0	0	22.34957	2.600368	0	800

3.3 Connectivity



Critial Facilities and Evacuation Routes

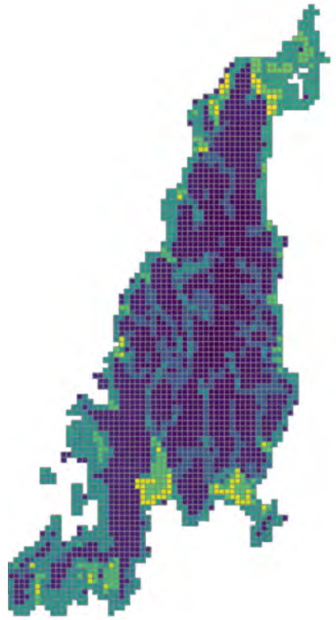
Critial Facilities

Evacuation Routes

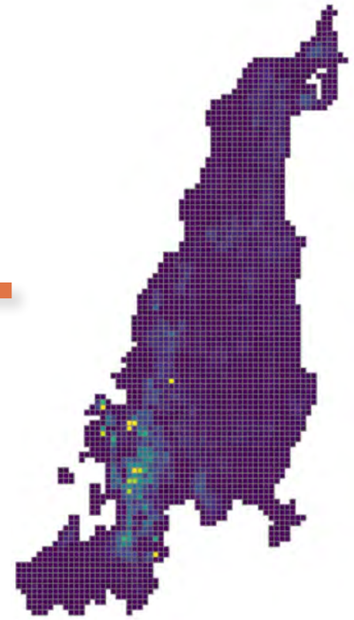
3.4 Weighted Summary of Three Scores

```
arcpy.AddField_management(inputLayer,"total_Score","DOUBLE")
enumerationOfRecords = arcpy.UpdateCursor(inputLayer)
for eachRecord in enumerationOfRecords:
    exposureScore = eachRecord.getValue("Expos_Score")
    sensitivityScore = eachRecord.getValue("Sensiti_Score")
    connectivityScore = eachRecord.getValue("Connect_Score")
    total_Score = exposureScore * 0.4 + sensitivityScore * 0.4 +
                  connectivityScore * 0.2
    eachRecord.setValue("total_Score",total_Score)
    enumerationOfRecords.updateRow(eachRecord)
del eachRecord
del enumerationOfRecords
```

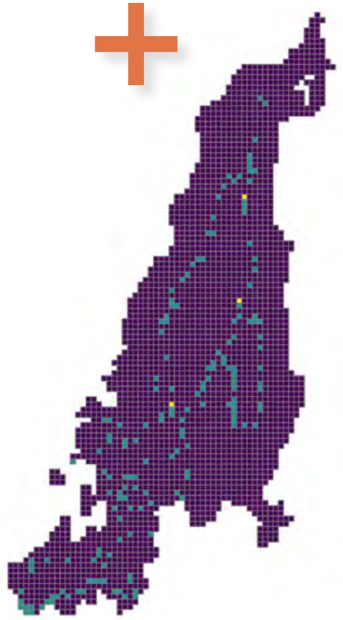
Weighted summary of three scores to compute the total social vulnerability score.



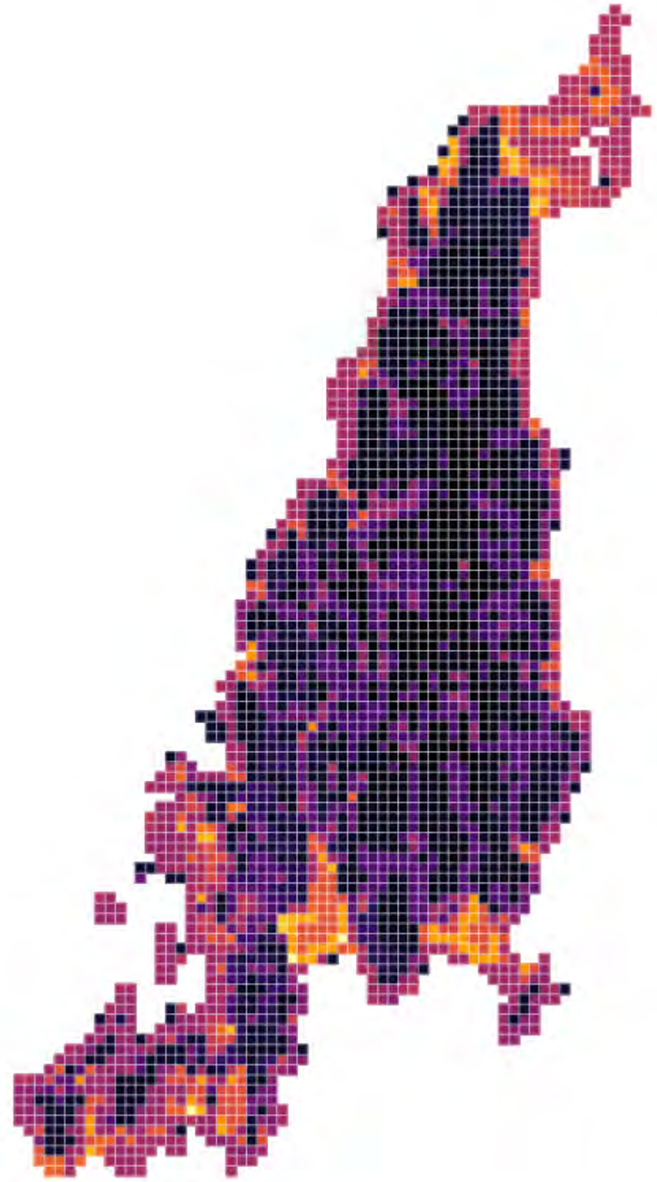
Exposure Score Map



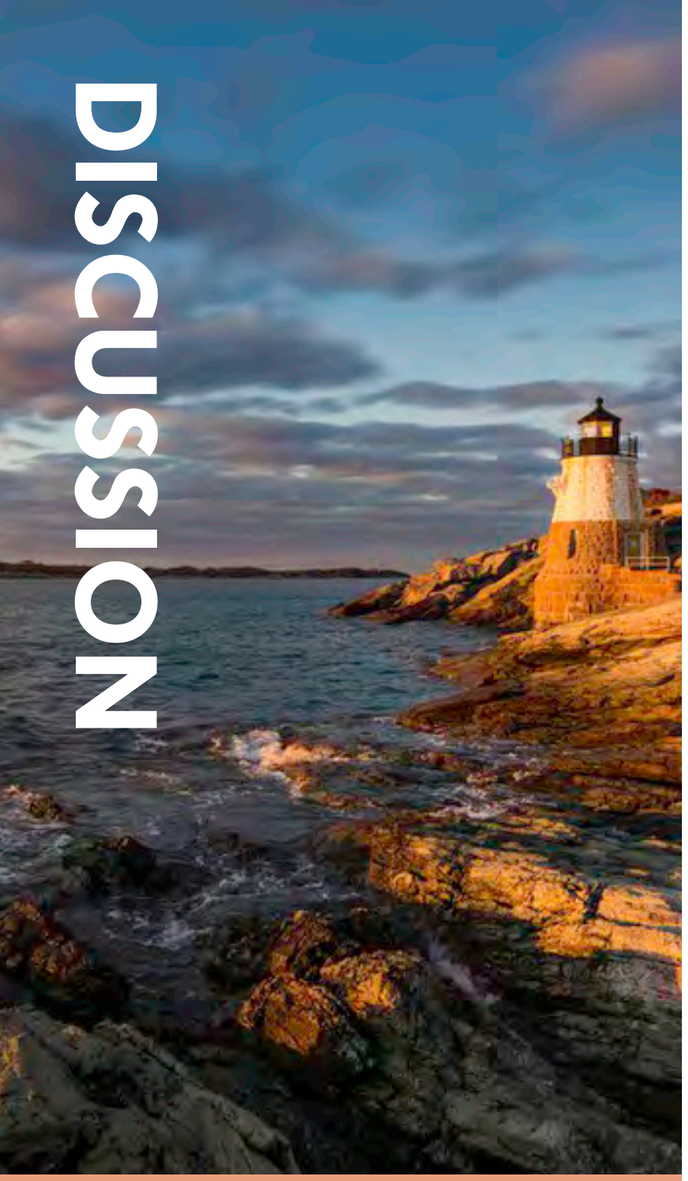
Sensitivity Score Map



Connectivity Score Map



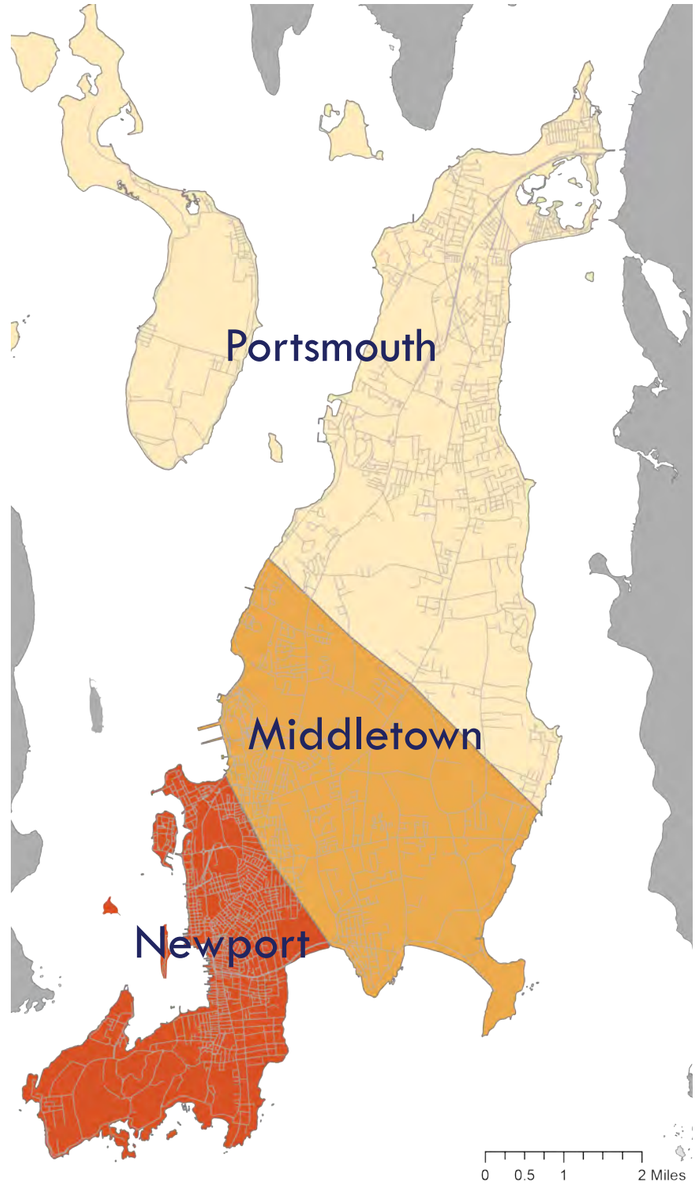
Social vulnerability score of Aquidneck Map



4. DISCUSSION

4.1 FINDINGS

4.2 LIMITATIONS & NEXT STEPS



Map of the three towns on Aquidneck Island

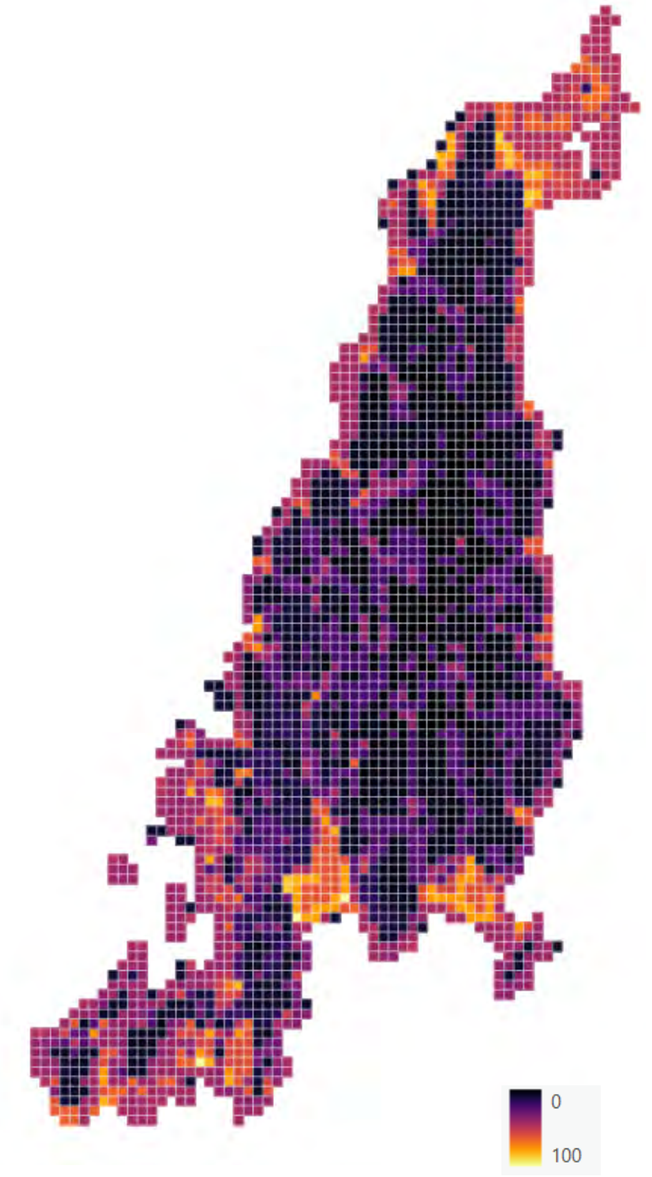
4.1 Findings

As shown on the social vulnerability map, the south end of Newport, the Newport Harbor, the north end of Portsmouth and the areas around the Atlantic Beach and Easton's Pond Reservoir and Easton Bay are most vulnerable when exposed to SLR related climate changes and natural hazards.

The north part of the Newport is the most vulnerable area in terms of socio economic sensitivity to the hazards. The area concentrates large underrepresented groups with low income, low house value, high racial minorities and low educational attainment. In terms of exposure, the storm surge flooding remains one big issues for many of the costal communities. Inland flooding and coastal soil erosion also threatens the reservoir near the sea.

4.2 LIMITATIONS & NEXT STEPS

Alough the spatial analysis model offer some insights on the social vulnerability of the Aquidneck Island. There are several limitations related to the application of this tools used in this project. Firstly, it is difficult to identify the most proper weights of the indicators and index. More expertise from the local communities and planning agengies should be involved in the next step. Second, multicollinearity might exist between the socio-economic indicators. In this study, indicators like number of people below poverty, educational attainment and median household income are also included. These indicators are closely correlated and might add redundancies to the model and exafferate certain vulnerability patterns. In the next steps, regression model should be run to test statistical significance of each indicator and multicollinearity should be avoided. Third, instead of using block group level average median household income data and house value data, more disaggregated housing data should be added into the model.



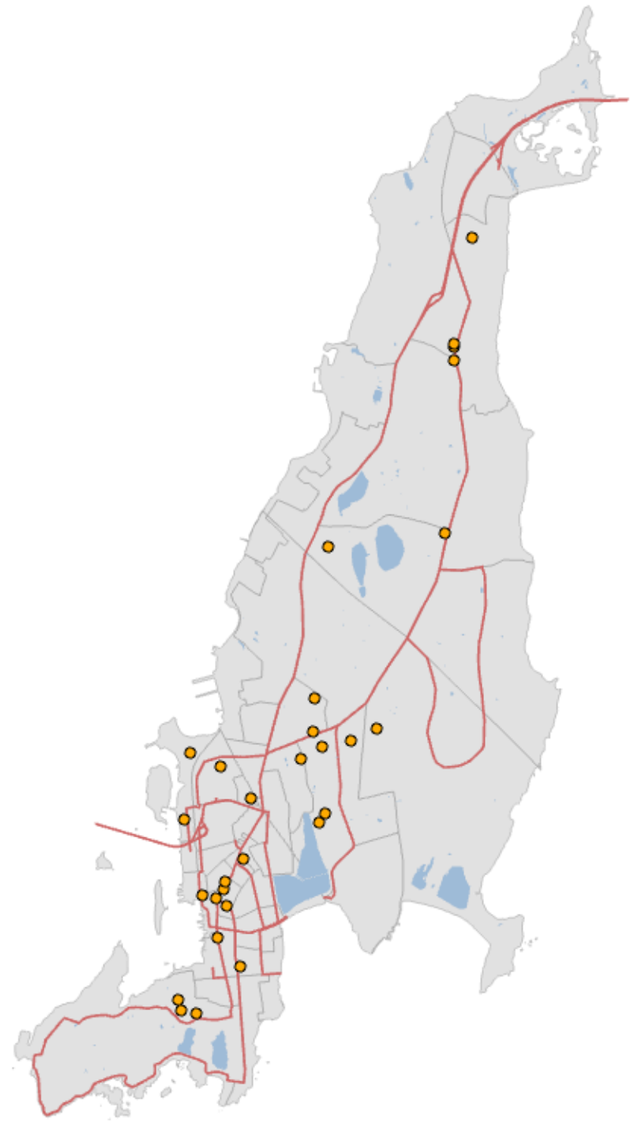
Social vulnerability score of Aquidneck Map

5. APPENDIX

5.1 DATA SOURCES

5.2 REFERENCE

5.3 CODE



Input data layers (pars of)

5.1 Data Sources

- “Rhode Island Geographic Information System.” Accessed December 18, 2020. https://www.rigis.org/.
- “STORMTOOLS – RI Shoreline Change Special Area Management Plan.” Accessed December 18, 2020. https://www.beachsamp.org/stormtools/.
- Google Developers. “WorldPop Global Project Population Data: Estimated Residential Population per 100x100m Grid Square.” Accessed December 18, 2020. https://developers.google.com/earth-engine/datasets/catalog/WorldPop_GP_100m_pop.

5.2 Reference

- Aksha, Sanam K., Luke Juran, Lynn M. Resler, and Yang Zhang. “An Analysis of Social Vulnerability to Natural Hazards in Nepal Using a Modified Social Vulnerability Index.” International Journal of Disaster Risk Science 10, no. 1 (March 2019): 103–16. https://doi.org/10.1007/s13753-018-0192-7.
- SeaLevelRise.org. “Rhode Island’s Sea Level Is Rising.” Sea Level Rise. Accessed October 13, 2020. https://sealevelrise.org/states/rhode-island/.
- Slocum, Terra A., Robert B. Kessler McMaster, and Hugh H. Howard. Thematic Cartography and Geovisualization. 3rd ed. Upper Saddle River, NJ: Pearson, 2009.
- “Social Vulnerability.” In Wikipedia, September 12, 2020. https://en.wikipedia.org/w/index.php?title=Social_vulnerability&oldid=978034043.
- Wood, Nathan J., Christopher G. Burton, and Susan L. Cutter. “Community Variations in Social Vulnerability to Cascadia-Related Tsunamis in the U.S. Pacific Northwest.” Natural Hazards 52, no. 2 (February 2010): 369–89. https://doi.org/10.1007/s11069-009-9376-1.

5.3 Code

```
"""
THIS SCRIPT CONDUCT A SERIES OF OPERATIONS TO EVALUATE THE SOCIAL VULNERABILITY OF AQUIDNECK ISLAND IN FACE OF SEA LEVEL RISING.
IN THIS PART SOCIO ECONOMIC DATA (CENSUS DATA) WILL BE ADDED AND PROCESSED IN THE MODEL
To create an ArcGIS Pro script tool for this script, do the following.
1   In Catalog > Toolboxes, select an existing toolbox or create a new one.
2   Right-click on the entry for this toolbox in ArcToolbox, and use New > Script to open a dialog box.
3   In this dialog box, use Label to name the tool being created and Script File to specify its .py file.
        LABEL/NAME          DATA TYPE       DIRECTION       DEPENDENCY
        block group?        FEATURE LAYER    INPUT
        population?         Raster LAYER     INPUT
        output?             SHAPEFILE        OUTPUT
4   To later revise any of this, right-click to the tool's name and select Properties.
"""

# Import external modules
import sys, os, string, math, arcpy, traceback, numpy

# Allow output to overwite any existing grid of the same name
arcpy.env.overwriteOutput = True

# If Spatial Analyst license is available, check it out
if arcpy.CheckExtension("spatial") == "Available":
    arcpy.CheckOutExtension("spatial")

    try:
    # Set local variables
    blockGroup = arcpy.GetParameterAsText(0)
    arcpy.AddMessage('\n'+'the block group layer is calld ' + blockGroup)
    population = arcpy.GetParameterAsText(1)
    arcpy.AddMessage('\n'+'the 100*100 meter population raster layer is calld ' + population)
    outputFile = arcpy.GetParameterAsText(2)
    arcpy.AddMessage('\n'+'the output file is calld ' + outputFile)

    # Adding in Socio-economic factors

    ## STEP 1: Execute GridIndexFeatures
    arcpy.GridIndexFeatures_cartography("Grid200m",blockGroup, "", "", "",
                                        "200 meters", "200 meters")

    ## STEP 2: Calculate zonal sum population of each cell
    arcpy.sa.ZonalStatisticsAsTable("Grid200m","PageNumber",population,"ZonalStable","DATA","SUM")

    ## STEP 3: Join the population table to the Grid shapefile
    joinedGrid200m = arcpy.AddJoin_management("Grid200m","PageNumber","ZonalStable","PageNumber","KEEP_ALL")

    ## STEP 4: Caluculate zonal sum population of each block group
    arcpy.sa.ZonalStatisticsAsTable(blockGroup,"GEOID",population,"ZonalStable2","DATA","SUM")

```

```
## STEP 5: Join the population table to the block group feature class
joinedBlockGroup = arcpy.AddJoin_management(blockGroup,"GEOID","ZonalStable2","GEOID","KEEP_ALL")


## STEP 6: Delete fields in the joined Block Group feature class that are no longer useful
desc = arcpy.Describe(joinedBlockGroup)
# arcpy.AddMessage(desc.datasetType)
#### 6-0: Create a feature layer
arcpy.CopyFeatures_management(joinedBlockGroup,"joinedBlockGroupLayer")
#### 6-1: Create a list of all the fields in the joined Block Group Layer
fieldList = arcpy.ListFields("joinedBlockGroupLayer")
#### 6-2: Creat a list to store the field names that we want to delete
toDelete = ["ZonalStable2_OBJECTID","ZonalStable2_GEOID","ZonalStable2_ZONE_CODE","ZonalStable2_COUNT","ZonalStable2_AREA"]
#### 6-3: Execute DeleteField_management
cleanBlockGroup = arcpy.DeleteField_management("joinedBlockGroupLayer",toDelete)


## STEP 7: Rename the fields in the attribute table of the joined Block Group layer
#### 7-1: Create a list of all the fields
tempList = arcpy.ListFields(cleanBlockGroup)
#### 7-2: Loop through all the fields
for field in tempList:
    # do not change those required fields
    if field.required == True:
        continue
    # Split field name at _ symbol
    fieldNames = field.name.split("_")
    # Delete the sufix
    del(fieldNames[0:1])
    # Set the new field value equal to the old one
    expression = "!"+field.name+"!"
    if len(fieldNames) == 1:
        if fieldNames[0] == "SUM":
            # add a new field using the same properties as the original field
            arcpy.AddField_management(cleanBlockGroup, "POP", field.type)
            # calculate the values of the new field
            # set it to be equal to the old field
            arcpy.CalculateField_management(cleanBlockGroup, "POP", expression)
            # delete the old fields
            arcpy.DeleteField_management(cleanBlockGroup, field.name)
            continue
        if field.type == "Integer":
            arcpy.AddField_management(cleanBlockGroup, fieldNames[0], "LONG",20)
            arcpy.CalculateField_management(cleanBlockGroup, fieldNames[0], expression)
            arcpy.DeleteField_management(cleanBlockGroup, field.name)
        elif field.type == "String":
            arcpy.AddField_management(cleanBlockGroup, fieldNames[0], "TEXT",30)
            arcpy.CalculateField_management(cleanBlockGroup, fieldNames[0], expression)
            arcpy.DeleteField_management(cleanBlockGroup, field.name)
        else:
            arcpy.AddField_management(cleanBlockGroup, fieldNames[0], field.type)
            arcpy.CalculateField_management(cleanBlockGroup, fieldNames[0], expression)
            arcpy.DeleteField_management(cleanBlockGroup, field.name)
    else:
        newFieldNames = fieldNames[0]+"_"+fieldNames[1]
        arcpy.AddField_management(cleanBlockGroup, newFieldNames, field.type)
        arcpy.CalculateField_management(cleanBlockGroup, newFieldNames, expression)
        arcpy.DeleteField_management(cleanBlockGroup, field.name)
#### 7-3: Review the new fields
# arcpy.AddMessage("\n")
# arcpy.AddMessage("This is the final fields of cleanBlockGroup: ")
# for newField in arcpy.ListFields(cleanBlockGroup):
#     arcpy.AddMessage(newField.name)


## STEP 8: Delete fields in the Grid shapefile that are no longer useful
desc = arcpy.Describe(joinedGrid200m)
#### 8-0: Create a feature layer for the Grid shapefile
# arcpy.AddMessage(desc.datasetType)
arcpy.CopyFeatures_management(joinedGrid200m,"joinedGrid200mLayer")
#### 8-1: Create a list of all the fields in the Grid Layer
fieldList = arcpy.ListFields("joinedGrid200mLayer")
#### 8-2: Creat an empty list to store the field names that we want to delete
toDelete = []
#### 8-3: Create a list to store the fields that we want to keep
theFieldWeWant = ["Grid200m_PageNumber","ZonalStable_SUM"]
#### 8-4: Loop through all the fields and delte the ones that are not required nor needed
for field in fieldList:
    # arcpy.AddMessage(field.name)
    if not field.name in theFieldWeWant and not field.required:
        toDelete.append(field.name)
# arcpy.AddMessage("This list is what we want to delete: ")
# arcpy.AddMessage(toDelete)
# arcpy.AddMessage("This list is what we want to save: ")
# arcpy.AddMessage(theFieldWeWant)
#### 8-5: Execute DeleteField_management
cleanGridLayer = arcpy.DeleteField_management("joinedGrid200mLayer",toDelete)
#### 8-6: Review the new fields
# arcpy.AddMessage("\n")
# arcpy.AddMessage("This is the final fields of cleanGridLayer: ")
# for newField in arcpy.ListFields(cleanGridLayer):
#     arcpy.AddMessage(newField.name)


## STEP 9: Rename the fields in the attribute table of the clean Grid Layer
#### 9-1: Create a list of all the fields
tempList = arcpy.ListFields(cleanGridLayer)
#### 9-2: Loop through all the fields
for field in tempList:
    # do not change those required fields
    if field.required == True:
        # arcpy.AddMessage("\n"+"This field is required "+ field.name)
        continue
    # Split field name at _ symbol
    # arcpy.AddMessage("This field will be renamed "+ field.name)
    # arcpy.AddMessage("The field type is "+ field.type)
    fieldNames = field.name.split("_")
    # Delete the sufix
    del(fieldNames[0:1])
    # Set the new field value equal to the old one
    expression = "!"+field.name+"!"
    if fieldNames[0] == "SUM":
        # add a new field using the same properties as the original field
        arcpy.AddField_management(cleanGridLayer, "grdPOP", field.type)
        # calculate the values of the new field
        # set it to be equal to the old field
        arcpy.CalculateField_management(cleanGridLayer, "grdPOP", expression)
        # delete the old fields
        arcpy.DeleteField_management(cleanGridLayer, field.name)
        continue
    if field.type == "Integer":
        arcpy.AddField_management(cleanGridLayer, fieldNames[0], "LONG",20)
        arcpy.CalculateField_management(cleanGridLayer, fieldNames[0], expression)
        arcpy.DeleteField_management(cleanGridLayer, field.name)
#### 9-3: Review the new fields
# arcpy.AddMessage("\n")
# arcpy.AddMessage("This is the final fields of cleanGridLayer: ")
# for newField in arcpy.ListFields(cleanGridLayer):
#     arcpy.AddMessage(newField.name)


## STEP 10: Spatial join the demographic attributes (in the block group shapefile) to the Grid layer
    # Want to join block group to the grid and calculate the mean census data
    # for each cell
    # Output will be the target features, states, with a mean city population field (mcp)
#### 10-1: Create a new fieldmappings and add the two input feature classes.
fieldmappings = arcpy.FieldMappings()
fieldmappings.addTable(cleanGridLayer)
fieldmappings.addTable(cleanBlockGroup)
```

```
        #### 10-2: Create a list of the fields that we want to join to the Grid layer
        spatialJoinFieldList = ["totPop","MedHHInc","MedHomVal","Pct_Oldand","Pct_MinRac","Pct_Bachel","Pct_PoverB","POP"]
        #### 10-3: Renew the fieldmap for each field that we want to join to the Grid layer
            # First get the fieldmap from fieldmapping. Each is a field in the cleanBlockGroup feature class.
            # The output will have the cells with the attributes of the block groups.
            # Setting the field's merge rule to mean will aggregate the values for all of the block groups that
            # each cell intersect with into an average value.
            # The field is also renamed to be more appropriate
            # for the output.
        for eachField in spatialJoinFieldList:
            # Get each field
            FieldIndex = fieldmappings.findFieldMapIndex(eachField)
            fieldmap = fieldmappings.getFieldMap(FieldIndex)
            # Get the output field's properties as a field object
            field = fieldmap.outputField
            # Rename the field and pass the updated field object back into the field map
            field.name = "M_" + eachField
            field.aliasName = "M_" + eachField
            fieldmap.outputField = field
            # Set the merge rule to mean and then replace the old fieldmap in the mappings object
            # with the updated one
            fieldmap.mergeRule = "mean"
            fieldmappings.replaceFieldMap(FieldIndex, fieldmap)
        #### 10-4: Delete fields that are no longer applicable, such as OBJECTID and GEOID of the block groups
        # fieldmappings.removeFieldMap(fieldmappings.findFieldMapIndex("cleanBlockGroup_OBJECTID"))
        fieldmappings.removeFieldMap(fieldmappings.findFieldMapIndex("GEOID"))
        #### 10-5: Run the Spatial Join tool
        arcpy.SpatialJoin_analysis(cleanGridLayer,cleanBlockGroup,"GridWithCEN","JOIN_ONE_TO_ONE","KEEP_ALL",fieldmappings,"INTERSECT")


        ## STEP 11: Delete, rename the fields of the spatial-joined grid.
        toDelete = ["Join_Count","TARGET_FID"]
        GridWithCEN = arcpy.DeleteField_management("GridWithCEN",toDelete)


        ## STEP 12: Create new fields and populate the values for the new field
        #### 12-1: Create new fields
        # arcpy.AddField_management(GridWithCEN,"Pct_POP","DOUBLE", 20, 8)
        arcpy.AddField_management(GridWithCEN,"Old_Yng","DOUBLE", 20, 8)
        arcpy.AddField_management(GridWithCEN,"highEdu","DOUBLE", 20, 8)
        arcpy.AddField_management(GridWithCEN,"minRace","DOUBLE", 20, 8)
        arcpy.AddField_management(GridWithCEN,"thePoor","DOUBLE", 20, 8)
        #### 12-2: Create new fields
        enumerationOfRecords = arcpy.UpdateCursor(GridWithCEN)
        for nextRecord in enumerationOfRecords:
            gridPopulation = nextRecord.getValue("grdPOP")
            blockGroupPopulation = nextRecord.getValue("M_POP")
            Pct_OldandYoung = nextRecord.getValue("M_Pct_Oldand")
            Pct_MinorRace = nextRecord.getValue("M_Pct_MinRac")
            Pct_BachelorAndAbove = nextRecord.getValue("M_Pct_Bachel")
            Pct_BelowPovertyLine = nextRecord.getValue("M_Pct_PoverB")
            Old_Yng = gridPopulation * Pct_OldandYoung
            highEdu = gridPopulation * Pct_BachelorAndAbove
            minRace = gridPopulation * Pct_MinorRace
            thePoor = gridPopulation * Pct_BelowPovertyLine
            nextRecord.setValue("Old_Yng",Old_Yng)
            nextRecord.setValue("highEdu",highEdu)
            nextRecord.setValue("minRace",minRace)
            nextRecord.setValue("thePoor",thePoor)
            enumerationOfRecords.updateRow(nextRecord)
        # Add a blank line at the bottom
        arcpy.AddMessage('\n')
        # Update the records
        del nextRecord
```

```
            "highEdu","minRace","thePoor"]
        New_nameOfFieldList = ["N_MedHHInc","N_MedHomVal","N_OldYng",
            "N_highEdu","N_minRace","N_thePoor"]
        for newName in New_nameOfFieldList:
            arcpy.AddField_management(GridWithCEN,newName,"DOUBLE",5,8)
        for nameOfValueField in nameOfFieldList:
            # Convert the attribute table to Numpy Array
            valueFieldArray = arcpy.da.TableToNumPyArray(GridWithCEN,
                                    nameOfValueField)
            minValue = numpy.min(valueFieldArray[nameOfValueField])
            maxValue = numpy.max(valueFieldArray[nameOfValueField])
            enumerationOfRecords1 = arcpy.UpdateCursor(GridWithCEN)
            for nextRecord in enumerationOfRecords1:
                OldValue = nextRecord.getValue(nameOfValueField)
                normalizedValue = (OldValue - minValue) / (maxValue - minValue) * 100
                nextRecord.setValue(New_nameOfFieldList[nameOfFieldList.index(nameOfValueField)],
                                    normalizedValue)
                enumerationOfRecords1.updateRow(nextRecord)
            arcpy.AddMessage("\n")
            del nextRecord
            del enumerationOfRecords1


        ## STEP 14: Clean the output shapefile's attribute table
        toDelete = ["M_totPop","M_MedHHInc","M_MedHomVal","M_Pct_Oldand","M_Pct_MinRac","M_Pct_Bache
        arcpy.DeleteField_management(GridWithCEN,toDelete)


        ## STEP 15: Save the output to export
        arcpy.CopyFeatures_management(GridWithCEN,outputFile)


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

    # Check in Spatial Analyst extension license
    arcpy.CheckInExtension("spatial")
    arcpy.AddMessage("\n"+"Success!")
else:
    arcpy.AddMessage ( "Spatial Analyst license is " + arcpy.CheckExtension("spatial") )
```

```
"""
THIS SCRIPT CONDUCT A SERIES OF OPERATIONS TO EVALUATE THE SOCIAL VULNERABILITY OF AQUIDNECK ISLAND IN FACE OF SEA LEVEL RISING.
IN THIS PART ENVIRONMENTAL FACTORS (ELEVATION, INUNDATION DEPTH, PROXIMITY TO PONDS AND RIVERS) WILL BE PROCESSED IN THE MODEL
To create an ArcGIS Pro script tool for this script, do the following.
1   In Catalog > Toolboxes, select an existing toolbox or create a new one.
2   Right-click on the entry for this toolbox in ArcToolbox, and use New > Script to open a dialog box.
3   In this dialog box, use Label to name the tool being created and Script File to specify its .py file.
        LABEL/NAME          DATA TYPE       DIRECTION       DEPENDENCY
        input grid?         FEATURE LAYER   INPUT
        elevation?          RASTER LAYER    INPUT
        output?             SHAPEFILE       OUTPUT
4   To later revise any of this, right-click to the tool's name and select Properties.
"""

# Import external modules
import sys, os, string, math, arcpy, traceback, numpy
from arcpy import env
from arcpy.sa import *

# Allow output to overwite any existing grid of the same name
arcpy.env.overwriteOutput = True

# If Spatial Analyst license is available, check it out
if arcpy.CheckExtension("spatial") == "Available":
    arcpy.CheckOutExtension("spatial")

    try:
        # Set local variables
        GridWithCEN = arcpy.GetParameterAsText(0)
        arcpy.AddMessage('\n'+'the input grid layer is called ' + GridWithCEN)
        elevation = arcpy.GetParameterAsText(1)
        arcpy.AddMessage('\n'+'the elevation raster layer is called ' + elevation)
        inundationDepth = arcpy.GetParameterAsText(2)
        arcpy.AddMessage('\n'+'the inundation raster of 100 year strom surge is called ' + inundationDepth)
        lakesPonds = arcpy.GetParameterAsText(3)
        arcpy.AddMessage('\n'+'the lakes and ponds feature layer is called ' + lakesPonds)
        riverStreams = arcpy.GetParameterAsText(4)
        arcpy.AddMessage('\n'+'the rivers and streams feature layer is called ' + riverStreams)
        outputFile = arcpy.GetParameterAsText(5)
        arcpy.AddMessage('\n'+'the output file is called ' + outputFile)

        # Adding in Environmental factors

        ## STEP 1: Reclassify the elevation cells where the value is lower than 0 to 0
        outputRASTER = arcpy.sa.Reclassify(elevation,"VALUE",RemapRange([[-3000,0,"NODATA"]]))

        ## STEP 2: Calculate zonal mean elevation of each cell
        arcpy.sa.ZonalStatisticsAsTable(GridWithCEN,"PageNumber",outputRASTER,"elevationTable","DATA","MEAN")

        ## STEP 3: Join the elevation table to the Grid shapefile
        joinedGridWithCEN = arcpy.AddJoin_management(GridWithCEN,"PageNumber","elevationTable","PageNumber","KEEP_ALL")

        ## STEP 4: Calculate zonal sum inundation depth of each cell
        arcpy.sa.ZonalStatisticsAsTable(joinedGridWithCEN,"GridWithCEN.PageNumber",inundationDepth,"inundationTable","DATA","MEAN")

        ## STEP 5: Join the inundation depth table to the Grid shapefile
        joinedGridWithCEN1 = arcpy.AddJoin_management(joinedGridWithCEN,"GridWithCEN.PageNumber","inundationTable","VALUE","KEEP_ALL")


        ## STEP 6: Delete fields in the attribute table that are no longer useful and rename some fields

        #### 6-1: Delete unnecessary fields
        arcpy.CopyFeatures_management(joinedGridWithCEN1,"joinedGridWithCEN1Layer")
        toDelete = ["GridWithCEN_Shape_Leng",
        "GridWithCEN_Shape_Area",
        "elevationTable_OBJECTID",
        "elevationTable_PageNumber",
        "elevationTable_COUNT",
        "elevationTable_AREA",
        "inundationTable_OBJECTID",
        "inundationTable_VALUE",
        "inundationTable_COUNT",
        "inundationTable_AREA"]
        joinedGridWithCEN2 = arcpy.DeleteField_management("joinedGridWithCEN1Layer",toDelete)

        #### 6-2: Rename the fields
        fieldList = arcpy.ListFields(joinedGridWithCEN2)
        for eachField in fieldList:
            if eachField.required:
                # arcpy.AddMessage("This field is required " + eachField.name)
                # arcpy.AddMessage("This field'S TYPE is " + eachField.type)
                continue
            else:
                fieldNames = eachField.name.split("_")
                expression = "!" + eachField.name + "!"
                if fieldNames[0] == "elevationTable":
                    del(fieldNames[0])
                    newName = "M_elevation"
                    arcpy.AddField_management(joinedGridWithCEN2,newName,"DOUBLE")
                    arcpy.CalculateField_management(joinedGridWithCEN2,newName,expression)
                    arcpy.DeleteField_management(joinedGridWithCEN2,eachField.name)
                elif fieldNames[0] == "inundationTable":
                    del(fieldNames[0])
                    newName = "M_inundation"
                    arcpy.AddField_management(joinedGridWithCEN2,newName,"DOUBLE")
                    arcpy.CalculateField_management(joinedGridWithCEN2,newName,expression)
                    arcpy.DeleteField_management(joinedGridWithCEN2,eachField.name)
                else:
                    del(fieldNames[0])
                    if len(fieldNames) == 1:
                        newName = fieldNames[0]
                        if eachField.type == "Integer":
                            newType = ["LONG",20]
                            arcpy.AddField_management(joinedGridWithCEN2,newName,newType[0]
                            arcpy.CalculateField_management(joinedGridWithCEN2,newName,expr
                            arcpy.DeleteField_management(joinedGridWithCEN2,eachField.name)
                        else:
                            arcpy.AddField_management(joinedGridWithCEN2,newName,eachField.
                            arcpy.CalculateField_management(joinedGridWithCEN2,newName,expr
                            arcpy.DeleteField_management(joinedGridWithCEN2,eachField.name)
                    else:
                        newName = fieldNames[0] + fieldNames[1]
                        arcpy.AddField_management(joinedGridWithCEN2,newName,eachField.type
                        arcpy.CalculateField_management(joinedGridWithCEN2,newName,expressi
                        arcpy.DeleteField_management(joinedGridWithCEN2,eachField.name)


    ## STEP 7: Buffer areas of impact around lakes and streams
    arcpy.Buffer_analysis(lakesPonds, "bufferLakesPonds", "15 Meters", "FULL", "FLAT",
                          "ALL")
```

```
        arcpy.Buffer_analysis(riverStreams, "bufferRiverStreams", "5 Meters", "FULL", "FLAT
                              "ALL")


        ## STEP 8: Spatial join the buffers to the grid layer
        joinedGridWithCEN3 = arcpy.SpatialJoin_analysis(joinedGridWithCEN2,"bufferLakesPond
                                 "joinedGridWithCEN3","JOIN_ONE_TO_ONE","KEEP_ALL","#","INTE
        joinedGridWithCEN4 = arcpy.SpatialJoin_analysis(joinedGridWithCEN3,"bufferRiverStre
                                 "joinedGridWithCEN4","JOIN_ONE_TO_ONE","KEEP_ALL","#","INTE


        ## STEP 9: Delete/Rename fields
        toDelete = ["Shape_Length_1","Shape_Area_1","Shape_Length_12","Shape_Area_12",'TARG
        arcpy.DeleteField_management(joinedGridWithCEN4,toDelete)
        fieldList1 = arcpy.ListFields(joinedGridWithCEN4)
        for eachField in fieldList1:
            if eachField.name == "Join_Count":
                newName = "ifLakesPonds"
                expression = "!"+eachField.name+"!"
                arcpy.AddField_management(joinedGridWithCEN4,newName,"SHORT")
                arcpy.CalculateField_management(joinedGridWithCEN4,newName,expression)
                arcpy.DeleteField_management(joinedGridWithCEN4,eachField.name)
            if eachField.name == "Join_Count_1":
                newName = "ifStrmsRivs"
                expression = "!"+eachField.name+"!"
                arcpy.AddField_management(joinedGridWithCEN4,newName,"SHORT")
                arcpy.CalculateField_management(joinedGridWithCEN4,newName,expression)
                arcpy.DeleteField_management(joinedGridWithCEN4,eachField.name)


        ## STEP 10: Normalize the values of fields
        # fieldList = arcpy.ListFields(joinedGridWithCEN4)
        # for field in fieldList:
        #     arcpy.AddMessage("This is the name of the field: " + field.name)

        arcpy.AddField_management(joinedGridWithCEN4,"N_grdPOP","DOUBLE",5,8)
        # arcpy.AddField_management(joinedGridWithCEN4,"N_elevation","DOUBLE",5,8)
        # arcpy.AddField_management(joinedGridWithCEN4,"N_inundation","DOUBLE",5,8)

        valueFieldArray = arcpy.da.TableToNumPyArray(joinedGridWithCEN4,"grdPOP")
        minValue = numpy.min(valueFieldArray["grdPOP"])
        maxValue = numpy.max(valueFieldArray["grdPOP"])
        enumerationOfRecords = arcpy.UpdateCursor(joinedGridWithCEN4)
        for nextRecord in enumerationOfRecords:
            OldValue = nextRecord.getValue("grdPOP")
            normalizedValue = (OldValue - minValue) / (maxValue - minValue) * 100
            nextRecord.setValue("N_grdPOP",normalizedValue)
            enumerationOfRecords.updateRow(nextRecord)
        arcpy.AddMessage("\n")
        del nextRecord
        del enumerationOfRecords

        # valueFieldArray = arcpy.da.TableToNumPyArray(joinedGridWithCEN4,"M_elevation")
        # minValue = numpy.min(valueFieldArray["M_elevation"])
        # maxValue = numpy.max(valueFieldArray["M_elevation"])
        # # arcpy.AddMessage("Maximum elevation: " + str(maxValue))
        # # arcpy.AddMessage("Minimum elevation: " + str(minValue))
        # enumerationOfRecords = arcpy.UpdateCursor(joinedGridWithCEN4)
        # for nextRecord in enumerationOfRecords:
        #     # arcpy.AddMessage(type(OldValue))
        #     # arcpy.AddMessage(OldValue)
        #     normalizedValue = (OldValue / 18410.375)  * 100
        #     nextRecord.setValue("N_elevation",normalizedValue)
        #     enumerationOfRecords.updateRow(nextRecord)
        # arcpy.AddMessage("\n")
        # del nextRecord
        # del enumerationOfRecords


        # valueFieldArray = arcpy.da.TableToNumPyArray(joinedGridWithCEN4,"M_inundation")
        # minValue = numpy.min(valueFieldArray["M_inundation"])
        # maxValue = numpy.max(valueFieldArray["M_inundation"])
        # enumerationOfRecords = arcpy.UpdateCursor(joinedGridWithCEN4)
        # for nextRecord in enumerationOfRecords:
        #     OldValue = nextRecord.getValue("M_inundation")
        #     normalizedValue = (OldValue / 17.322827) * 100
        #     nextRecord.setValue("N_inundation",normalizedValue)
        #     enumerationOfRecords.updateRow(nextRecord)
        # arcpy.AddMessage("\n")
        # del nextRecord
        # del enumerationOfRecords


        # toDelete = ["M_elevation","M_inundation"]
        # joinedGridWithCEN5 = arcpy.DeleteField_management(joinedGridWithCEN4,toDelete)


        arcpy.CopyFeatures_management(joinedGridWithCEN4,outputFile)
        # outputRASTER.save(outputFile)

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

    # Check in Spatial Analyst extension license
    arcpy.CheckInExtension("spatial")
    arcpy.AddMessage("\n"+"Success!")
else:
    arcpy.AddMessage ( "Spatial Analyst license is " + arcpy.CheckExtension("spatial") )
```

```
"""
THIS SCRIPT CONDUCT A SERIES OF OPERATIONS TO EVALUATE THE SOCIAL VULNERABILITY OF AQUIDNECK ISLAND IN FACE OF SEA LEVEL RISING.
IN THIS PART CRITICAL FACILITIES WILL BE ADDED AND PROCESSED IN THE MODEL
To create an ArcGIS Pro script tool for this script, do the following.
1   In Catalog > Toolboxes, select an existing toolbox or create a new one.
2   Right-click on the entry for this toolbox in ArcToolbox, and use New > Script to open a dialog box.
3   In this dialog box, use Label to name the tool being created and Script File to specify its .py file.
        LABEL/NAME          DATA TYPE       DIRECTION       DEPENDENCY
        block group?        FEATURE LAYER    INPUT
        population?         Raster LAYER     INPUT
        output?             SHAPEFILE        OUTPUT
4   To later revise any of this, right-click to the tool's name and select Properties.
"""

# Import external modules
import sys, os, string, math, arcpy, traceback, numpy

# Allow output to overwite any existing grid of the same name
arcpy.env.overwriteOutput = True

# If Spatial Analyst license is available, check it out
if arcpy.CheckExtension("spatial") == "Available":
    arcpy.CheckOutExtension("spatial")

    try:
        # Set local variables
        grid = arcpy.GetParameterAsText(0)
        arcpy.AddMessage('\n'+'the input grid layer is called ' + grid)
        criticalFacilities = arcpy.GetParameterAsText(1)
        arcpy.AddMessage('\n'+'the critical facilities feature layer is calld ' + criticalFacilities)
        evacuRoutes = arcpy.GetParameterAsText(2)
        arcpy.AddMessage('\n'+'the evacuRoutes feature layer is calld ' + evacuRoutes)
        outputFile = arcpy.GetParameterAsText(3)
        arcpy.AddMessage('\n'+'the output file is calld ' + outputFile)

        # Adding in critical facilities

        ## STEP 1: Spatil join the critical facilities to the grid layer
        joinedGrid = arcpy.SpatialJoin_analysis(grid,criticalFacilities,"joinedGrid","JOIN_ONE_TO_ONE","KEEP_ALL","#","INTERSECT")

        ## STEP 15: Save the output to export
        arcpy.CopyFeatures_management(joinedGrid,outputFile)


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

    # Check in Spatial Analyst extension license
    arcpy.CheckInExtension("spatial")
    arcpy.AddMessage("\n"+"Success!")
else:
    arcpy.AddMessage ( "Spatial Analyst license is " + arcpy.CheckExtension("spatial") )
```

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

# Allow output to overwite any existing grid of the same name
arcpy.env.overwriteOutput = True

# If Spatial Analyst license is available, check it out
if arcpy.CheckExtension("spatial") == "Available":
    arcpy.CheckOutExtension("spatial")

    try:
        # Set local variables
        inputFile = arcpy.GetParameterAsText(0)
        arcpy.AddMessage('\n'+'the input feature layer is called ' + inputFile)
        outputFile = arcpy.GetParameterAsText(1)
        arcpy.AddMessage('\n'+'the output file is called ' + outputFile)

        inputLayer = arcpy.CopyFeatures_management(inputFile,"inputLayer")
        fieldList = arcpy.ListFields(inputFile)
        for field in fieldList:
            arcpy.AddMessage(field.name)
            arcpy.AddMessage(field.required)

        arcpy.AddField_management(inputLayer,"hasPOI","SHORT")
        arcpy.AddField_management(inputLayer,"hasRoute","SHORT")

        enumerationOfRecords = arcpy.UpdateCursor(inputLayer)
        for eachRecord in enumerationOfRecords:
            if eachRecord.getValue("Join_Count") == 1:
                eachRecord.setValue("hasPOI",100)
                enumerationOfRecords.updateRow(eachRecord)
            if eachRecord.getValue("Join_Cou_1") == 1:
                eachRecord.setValue("hasRoute",100)
                enumerationOfRecords.updateRow(eachRecord)
        del eachRecord
        del enumerationOfRecords


        enumerationOfRecords = arcpy.UpdateCursor(inputLayer)
        for eachRecord in enumerationOfRecords:
            score1 = eachRecord.getValue("ifLakesPon")
            score2 = score1 * 100
            eachRecord.setValue("ifLakesPon",score2)
            enumerationOfRecords.updateRow(eachRecord)
            score3 = eachRecord.getValue("ifStrmsRiv")
            score4 = score3 * 100
            eachRecord.setValue("ifStrmsRiv",score4)
```

```
            enumerationOfRecords.updateRow(eachRecord)
        del eachRecord
        del enumerationOfRecords



        arcpy.AddField_management(inputLayer,"Expos_Score","DOUBLE")
        enumerationOfRecords = arcpy.UpdateCursor(inputLayer)
        for eachRecord in enumerationOfRecords:
            strom_s = eachRecord.getValue("N_inundat")
            proximityLake_s = eachRecord.getValue("ifLakesPon")
            proximityRiver_s = eachRecord.getValue("ifStrmsRiv")
            inundation_s = 100 - eachRecord.getValue("N_eleva")
            exposureScore = strom_s * 0.5 + proximityLake_s * 0.2
            + proximityRiver_s * 0.2 + inundation_s * 0.1
            eachRecord.setValue("Expos_Score",exposureScore)
            enumerationOfRecords.updateRow(eachRecord)
        del eachRecord
        del enumerationOfRecords


        arcpy.AddField_management(inputLayer,"Sensiti_Score","DOUBLE")
        enumerationOfRecords = arcpy.UpdateCursor(inputLayer)
        for eachRecord in enumerationOfRecords:
            population_s = eachRecord.getValue("N_grdPOP")
            poverty_s = eachRecord.getValue("NthePoor")
            minorRace_s = eachRecord.getValue("NminRace")
            vulnerableAge_s = eachRecord.getValue("NOldYng")
            education_s = 100 - eachRecord.getValue("NhighEdu")
            medHHInc_s = 100 - eachRecord.getValue("NMedHHInc")
            medHVal_s = 100 - eachRecord.getValue("NMedHomVa")
            sensitivityScore = population_s * 0.52 + poverty_s * 0.15
            + minorRace_s * 0.15 + vulnerableAge_s * 0.15
            + education_s * 0.01 + medHHInc_s * 0.01 + medHVal_s * 0.01
            eachRecord.setValue("Sensiti_Score",sensitivityScore)
            enumerationOfRecords.updateRow(eachRecord)
        del eachRecord
        del enumerationOfRecords


        arcpy.AddField_management(inputLayer,"Connect_Score","DOUBLE")
        enumerationOfRecords = arcpy.UpdateCursor(inputLayer)
        for eachRecord in enumerationOfRecords:
            critialFacility_s = eachRecord.getValue("hasPOI")
            evacuationRoute_s = eachRecord.getValue("hasRoute")
            connectivityScore = critialFacility_s * 0.5 +
                                evacuationRoute_s * 0.5
            eachRecord.setValue("Connect_Score",connectivityScore)
            enumerationOfRecords.updateRow(eachRecord)
        del eachRecord
        del enumerationOfRecords


        arcpy.AddField_management(inputLayer,"total_Score","DOUBLE")
        enumerationOfRecords = arcpy.UpdateCursor(inputLayer)
        for eachRecord in enumerationOfRecords:
            exposureScore = eachRecord.getValue("Expos_Score")
            sensitivityScore = eachRecord.getValue("Sensiti_Score")
            connectivityScore = eachRecord.getValue("Connect_Score")
            total_Score = exposureScore * 0.4 + sensitivityScore * 0.4 +
                          connectivityScore * 0.2
            eachRecord.setValue("total_Score",total_Score)
            enumerationOfRecords.updateRow(eachRecord)
        del eachRecord
        del enumerationOfRecords

        # !Expos_Scor! * 0.4  + !Sensiti_Sc! * 0.4 + !Connect_Sc! * 0.2


        arcpy.DeleteField_management(inputLayer,["Join_Count","Join_Cou_1","Shape_Le_1"])

        outputFile = arcpy.CopyFeatures_management(inputLayer,outputFile)

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

    # Check in Spatial Analyst extension license
    arcpy.CheckInExtension("spatial")
    arcpy.AddMessage("\n"+"Success!")
else:
    arcpy.AddMessage ( "Spatial Analyst license is " + arcpy.CheckExtension("spatial") )
```