The ArcPy tool described in the following document assigns risk of violent incident score to every school in Philadelphia based on nearby environmental factors. Schools with a higher risk score are more likely to see violent crime on their grounds over the course of a year. Risk score is based on either Geographically Weighted Regression or Ordinary Least Squares Regression estimates of the number of violent incidents which will take place at that school using the distance to nearby environmental factors as predictor variables. For example, one theory holds that schools which are near areas which see foot traffic at night have more violent incidents. Thus, distance to the nearest 24-hour convenience store is a predictor variable.

The code is not specific to schools, however, and can be used to calculate a risk score for any feature, as long as it has a field to predict and distance to at least one different feature is a predictor. The code eventually creates a python field calculator statement which corresponds to the regression formula below:

$$\bar{y}_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m$$

Where:

- $\bar{y}$ is the predicted number of violent incidents for school i.This is the "raw risk score" which the code creates, then assigns a ranking to get the risk score.
- β_0 is the intercept of the regression equation. In geographically weighted regression, every school is given a separate intercept, but in ordinary least squares regression, the intercept is universal to all schools. It is the number of violent incidents that would occur at the school if all of the predictor variables were set to zero.
- x_1 through x_m are the predictor variables. In this code the predictor variables are all distances to the nearest occurrence of a decision factor. For example, the distance to the nearest convenience store. In both ordinary least squares and geographically weighted regression, the values of the predictor variables are specific to each school.
- β_1 through β_m are coefficients representing how many more violent incidents a school experience for every unit change in the predictor variable, i.e. for every foot closer a given decision factor is to the school. These beta-coefficients are different for every school when using geographically weighted regression, but the same when using ordinary least squares.
- There is no error term ε here. The error term is the difference between the actual number of violent incidents at the school and the predicted number of violent incidents at the school. Adding it to the equation above would set the equation equal to the actual number of violent incidents, but we are only interested in the predicted number of incidents for determining risk score.

Script inputs:

1. The feature class you want to calculate risk score for
2. A set of decision factors you want to use to calculate risk
3. A field in the feature class which corresponds to what the risk ranking intends to predict. In other words, the risk of what?
4. A Boolean value indicating whether to calculate risk using geographically weighted regression. If set to "false" the script will default to ordinary least squares regression.
5. If the Boolean value noted above is set to false, the script will call for a list of Ordinary Least Squares

It should be noted that the script calculates beta-coefficients for geographically weighted regression, but does not for ordinary least squares regression. While it's standard practice for statisticians to run geographically weighted regression in ArcGIS, the program is not as effective in running ordinary least squares regression. The code contains functions that create fields equal to every variable in the regression equation, like so:

$$\bar{y}_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

RawRisk		Intercept		C1_Near1		Near1		C2_Near2		Near2	
172.954638		-34.320487		-0.007397		1078.958688		0.396076		574.802691	
148.446703	=	-13.604519	+ (	0.069525	x	2963.686974	) + (	-0.222808	x	252.025396	)
121.747348		-26.084834		0.067283		2951.275737		-0.178879		410.208643	
102.504434		-43.552925		0.076198		1593.324106		0.065037		150.740169	
97.469386		-17.115274		0.021353		2304.494867		0.102496		529.733745	

When using ordinary least squares regression, the coefficient fields

These fields are then exported to a new shapefile which in which another field is created that ranks that RawRisk scores from largest to smallest:

Rank		RawRisk
1		172.954638
2	=	148.446703
3		121.747348
4		102.504434
5		97.469386

Script outputs:

1. A new shapefile which contains the fields above
2. A .dbf file including some statistics from the Geographically Weighted Regression, not produced if using ordinary least squares regression.
3. A message to the user including which Near and Coefficient fields correspond to which desciion factor. If using ordinary least squares regression, the coefficients are listed as well. An example can be seen below:

```
Start Time: Wed Dec 16 19:34:25 2015
Running script AddRiskFields...
Crimes_2014 is Decision Factor Number 1


Convenience_Stores_1 is Decision Factor Number 2


DHS_OST_Programs is Decision Factor Number 3


The Weight for Decision Factor Number 1 is 0.17


The Weight for Decision Factor Number 2 is 0.19


The Weight for Decision Factor Number 3 is 0.08


...and the Intercept is 13.4
```

The full code, with inline commentary is below:

```
"""
```

```
THIS PERFORMS A REGRESSION ON A FEATURE TO CREATE A PREDICTION FIELD BASED ON
NEARBY FEATURES

To create an ArcToolbox tool with which to execute this script, do the
following.
1   In  ArcMap > Catalog > Toolboxes > My Toolboxes, either select an
existing toolbox
    or right-click on My Toolboxes and use New > Toolbox to create (then
rename) a new one.
2   Drag (or use ArcToolbox > Add Toolbox to add) this toolbox to ArcToolbox.
3   Right-click on the toolbox in ArcToolbox, and use Add > Script to open a
dialog box.
4   In this Add Script dialog box, use Label to name the tool being created,
and press Next.
5   In a new dialog box, browse to the .py file to be invoked by this tool,
and press Next.
6   In the next dialog box, specify the following inputS (using dropdown
menus wherever possible)
    before pressing OK or Finish.
        DISPLAY NAME                DATA TYPE      PROPERTY>DIRECTION>VALUE
        At Risk Feature             Shapefile      Input
        Decision Factors            Shapefile      Input
        Predict Field               Field          Input
        Geographically Weighted?    Boolean        Input
        OLS Weights and Intercept   Double         Input

        MULTIVALUE     OBTAINED FROM               TYPE
        No             (Leave Blank)               Required
        Yes            (Leave Blank)               Required
        No             At Risk Feature             Optional
        No             (Leave Blank)               Required
        Yes            (Leave Blank)               Optional

7   To later revise any of this, right-click to the tool's name and select
Properties.
"""
#Import Modules
import sys,os,string,traceback,arcpy

try:
    #Get feature for which you would like to assign a risk score
    Schools = arcpy.GetParameterAsText(0)
    #Get a list of the decision factors which influence the risk score
    #Use "GetParameter"  because Geographically Weighted Regression function
works
    #with geography objects rather than text
    Factors = arcpy.GetParameter(1)
    #Get the field which the risk score is intended to predict,
    #aka the dependent variable in our regressions.
    #This field can be left blank if using ordinary least squares regression
    PredictionField = arcpy.GetParameterAsText(2)
    #A boolean value that is true if the user would like to preform
    #Geographically Weighted Regression to calculate risk.
    #A value of false indicates the user intends to do OLS Regression
    GWRBool = arcpy.GetParameterAsText(3)
    #Optional list of beta-coefficents (a.k.a. weights) for OLS regression.
    #The first element of this list is the intercept
```

```
    #This field can be left blank if the user intendes to calculate risk
    #using Geographically Weighted Regression
    OLSWeightList = arcpy.GetParameterAsText(4)

    #AddNumField is a helper function
    #(to be used in other functions but not on its own)
    #It creates a field to hold numeric values,
    #and sets that field equal to a Python statement
    #RiskFeatures refers to the Feature Class the user
    #would like to assign a risk score
    def AddNumField (RiskFeatures,Newfieldname,CalculatorStatement):
        #All fields created using these fucntions will be Doubles with a
        #precision of 20 and a scale of 10
        arcpy.AddField_management(RiskFeatures,Newfieldname,"DOUBLE", 20, 10)
        #The field will be set to the CalculatorStatement, written in Python
        arcpy.CalculateField_management(RiskFeatures,Newfieldname,
CalculatorStatement,"PYTHON_9.3")

    #addnearfield adds a field to the RiskFeatures that is equal to
    #the distance to the nearest instance of Risk Factor X.
    #These are the predictor variables the regression uses to calculate risk
    def addnearfield(RiskFeatures,Factor,LoopCount):
        #Makes a the LoopCount integer  into a string
        #in order to use it in creating the field name
        Countstring = str(LoopCount)
        #Each field will be named "NearX" with X indicating the place
        # each factor lies in the list of factors (starting with 1)
        #So as to differentiate between each decision factor's Near Field
      newfieldname = "Near" + Countstring
        #The name of each Near Field will be added to a list
        # so it can be looped through when calculating risk
        Nearfields.append(newfieldname)
        #Run the near function to get the distance to
        # the nearest instance of a factor
        arcpy.Near_analysis(RiskFeatures,Factor)
        #Add a near field equal to the distance to the nearest instace of the
factor called NearX
        AddNumField(RiskFeatures,newfieldname,"!NEAR_DIST!")
        #The ArcGIS Near Function creates two fields that now are unnecessary
        #They must be deleted as they would prevent us
        # from looping through each factor if kept around.
        arcpy.DeleteField_management(RiskFeatures,"NEAR_DIST")
        arcpy.DeleteField_management(RiskFeatures,"NEAR_FID")

    #getweightsGWR creates intercept and the weights
    #(beta-coeffiecents) for each descision factor using
    #Geographically Weighted Regression Function
    def getweightsGWR(RiskFeatures,Nearlist, PredictField):
        #Name the shapefile that will be created
        #by using the geographically weighted regression function
        #This file will be delted later,
        #and it's relevant fields will be added to the RiskFeatures
      GWRShp = RiskFeatures[:-4] + "_GWR" + ".shp"
        #Dbf file detailing some statistics about the regression is created
        GWRDbf = GWRShp[:-4] + "_supp" + ".shp"
        #The Geographically Weighted Regression function takes multiple
        # values only if they are one text string seperated by semicolons
```

```
        ExplanitoryFields = ";".join(Nearfields)
          #Use Geographically Weighted Regression to create a new shapefile
          # which has values for the Near Field's beta-coefficents
          #Adaptive bandawith with 30 observations means the nearest 30 points
          # will be used to create the beta-coefficent with a localized effect
        arcpy.GeographicallyWeightedRegression_stats (RiskFeatures,
PredictField, ExplanitoryFields,
              GWRShp, "ADAPTIVE",  "BANDWIDTH PARAMETER", "#", "30","#", "#",
"#", "#", "#", "#")
          #Create an empty list that will be filled
          #with fields to drop from the regression shapefile
          DropList = []
          #Create a list of fields in the regression shapefile
          listOfFields= arcpy.ListFields(GWRShp)
          #Loop through this list, adding unwanted fields to the Droplist
          for nextFieldObject in listOfFields:
              #The fields we want are the coefficient  estimates,
              # which start with C, so they're not added to the list
              if nextFieldObject.name[0] == "C":
                  pass
              #Field indicating whether the results are due to
              #multicollinearity.Not useful for the script, but
              #begins with a "C" so it must be dropped
              #as an exception to the above line of code
              elif nextFieldObject.name == "Cond":
                  DropList.append(nextFieldObject.name)
              #We want to keep the intercept to use in
              #the calculation of raw risk score
              elif nextFieldObject.name == "Intercept":
                  pass
              #Neither FID nor Shape can be deleted from a feature's table
              elif nextFieldObject.name == "Shape":
                  pass
              elif nextFieldObject.name == "FID":
                  pass
              #Add all the other fields to the list of fields to drop
              else:
                  DropList.append(nextFieldObject.name)
          #Delete all unneccesary fields from the regression shp file
          arcpy.DeleteField_management(GWRShp,DropList)
          #Join the relevant fields to the RiskFeatures
          arcpy.JoinField_management(RiskFeatures,"FID",GWRShp,"FID")
          #Delete the regression shapefile
          arcpy.Delete_management(GWRShp)

      #getweightsOLS creates fields representing the intercept
      #as well as beta-coefficents for each predictor.
      def getweightOLS(Riskfeatures,WeightList):
          #The list of OLS weights is a multivalue parameter, and thus is read
          # as a text string where each element is separated by a semicolon
          #So splitting the text string at each semicolon gives a list
          # of weights that can be looped through
          ListWeights = WeightList.split(";")
          #Each field created by the function includes a number that
          #corresponds to which predictor's beta-coefficient it contains
          #And how many times the loop's code has been run
          LoopCount = 1
```

```
        #The first element of the OLS Weights list is the intercept
        InterceptW = ListWeights[0]
        #Create a new field named intercept and set it to
        #the first value of the OLS Weights List
        InterceptFieldName = "Intercept"
        AddNumField(Riskfeatures,InterceptFieldName,InterceptW)
        #The rest of the OLS Weights list are the
        #beta-coeffiecents for each predictor
        ListWeights2 = ListWeights[1:]
        #Loop through each of the list of beta-coefficents,
        # and create a new field for each one
        for OLSWeight in ListWeights2:
            #The field name includes a number that corresponds
            # to each predictor, must be in string form to use
            Countstring = str(LoopCount)
            #The field names are the same ones ArcGIS uses for
            #beta-coefficents in the Geographically Weighted Regression
Function
            #Each field is named "CX_NearX" where X is the
            #number responding to each predictor. "C" stands for coefficient.
            newfieldname2 = "C" + Countstring + "_Near" + Countstring
            #Create a new field set equal to the coefficent
            AddNumField(Riskfeatures,newfieldname2,OLSWeight)
            #Add 1 to the LoopCount to create a field for the next
            # sequential decision factor's beta-coefficient
            LoopCount = LoopCount + 1

    #rawscore creates a field which contains an estimation of the field
    # the user intends to predict using the regression equation
    def rawscore(Riskfeatures):
        #Name of the new field is RawRisk, meaning unranked
        RawField = "RawRisk"
        #Create an empty list of elements of the regression equation
        ExpressionElements = []
        #Keeps track of how many times the loop has cycled through,
        #which corresponds to what predictor is being added to the equation
        LoopCount = 1
        #Loop through every near field and create an expression
        # to be used to calculate RawRisk
        for Nearfield in Nearfields:
            #The LoopCount must be in string form to use
            Countstring = str(LoopCount)
            #Create an expression that reads "!NearX! * !CX_NearX!"
            #This is a Python expression corresponding to the predictor
            # value multiplied by its beta-coefficient
            Element = "!" + Nearfield + "!" + " " "*" + " " + "!" + "C" +
Countstring + "_" + Nearfield +"!"
            #Add the Python expression to the list of expressions
            ExpressionElements.append(Element)
            #Increase the LoopCount by one to create a python
            # expression for the next decision factor
            LoopCount = LoopCount + 1
        #Add the intercept to the list of expressions
        ExpressionElements.append("!Intercept!")
        #Create a single string out of every expression,
        # joined by the addition sign
        #The final expression reads:
```

```
        #"!NearX! * !CX_NearX! + "!NearX2! * !CX_NearX2!... + "!Intercept!"
        #Where the ellipses indicates that there is an expression
        # "!NearX! * !CX_NearX!" for every X predictor
        Expression = " + ".join(ExpressionElements)
        #Add a new field equal to the Python expression created in the line
above
        AddNumField(Riskfeatures,RawField,Expression)

    #the rank function takes a field, sorts it from largest to smallest,
    #and assigns an integer value based on its new sorted order
    def rank(Riskfeatures,fieldname):
            #Create the name of the final output shapefile
            RiskfeaturesRanked = Riskfeatures[:-4] + "_Ranked" + ".shp"
            #Create the name of the ranking field
            newfieldname = "Rank"
          #Sort the points by the fieldname attribute
            arcpy.Sort_management(Riskfeatures, RiskfeaturesRanked,
[[fieldname, "DESCENDING"]])
            #Create a new field to hold rank values
            arcpy.AddField_management(RiskfeaturesRanked, newfieldname,
"DOUBLE", 20, 5)
            # Create an enumeration of updatable records from
            #the shapefile's attribute table
            enumerationOfRecords = arcpy.UpdateCursor(RiskfeaturesRanked)
                  #Assign an initial value for the rank field,
            # that can be added to each time the loop is run
            rank = 1
            #Loop through every row of the shapefile's attribute table
            for nextRecord in enumerationOfRecords:
                #Set the value of the rank field to rank
                  nextRecord.setValue(newfieldname, rank)
                #Increase the rank by 1
                  rank = rank + 1
                #Go down a row in the attribute table,
                #to the newfieldname attribute with the next smallest value
                  enumerationOfRecords.updateRow(nextRecord)
            # Delete row and update cursor objects
            #to avoid locking attribute table
            del nextRecord
            del enumerationOfRecords

    #factorlist adds an message in the processing window showing
    #each decision factor and its corresponding number
    #This is to avoid confusion as to which fields
    #(with names like "Near1" and "Near2") correspond to which predictors
    def factorlist(ListofFactors):
        #Keeps track of how many times the loop has cycled through
        #which corresponds to what predictor the AddMessage line is about
      LoopCount = 1
        #Loop through the list of factors
      for F in ListofFactors:
            #Only strings can be used in the AddMessage function,
            # but the factors are geographic objects
            Fstring = F.name
            #The LoopCount must be in string form to use
            Countstring = str(LoopCount)
            #The message will read, for example:
```

```
            # "LiquorStores is Decision Factor Number 3"
            FMessage = Fstring + " is Decision Factor Number " + Countstring
            #Add the message
            arcpy.AddMessage(FMessage)
            #Add a space below the message for readability
            arcpy.AddMessage('\n')
            #Increase the LoopCount so as to match the correct number
            # to each decision factor
            LoopCount = LoopCount + 1

    #displayweightlist adds a message for each OLS weights
    # that tells the user which decision factor they correspond to
    def displayweightlist(WeightList):
        #Keeps track of how many times the loop has cycled through
        #which corresponds to what OLS beta-coefficient
        # the AddMessage line is about
        LoopCount = 1
        #The list of OLS weights is a multivalue parameter, and thus is
        # read as a text string where each element is separated by a
semicolon
        #So splitting the text string at each semicolon gives
        # a list of weights that can be looped through
        ListWeights = WeightList.split(";")
        #The first element of the weights list is the intercept,
        # so grab the rest of the list to loop through
        ListWeights2 = ListWeights[1:]
        for OLSWeight in ListWeights2:
            #Convert the LoopCount to a string in order to use it in
            #the AddMessage function
            Countstring = str(LoopCount)
            #The message reads, for example:
            # "The Weight for Decision Factor Number 4 is 0.16"
            WMessage = "The Weight for Decision Factor Number " + Countstring
+ " is " + OLSWeight
            arcpy.AddMessage(WMessage)
            #Add a space after the message for readability
            arcpy.AddMessage('\n')
            #Add one to the loopcount so as to add a message for the
            # next sequential decision factor
            LoopCount = LoopCount + 1
        #The Intercept is the first element of the OLS Weights List
        Intercept = ListWeights[0]
        #Add a message that reads, for example,\:
        # "...and the Intercept is 13.5"
        IMessage = "...and the Intercept is " + Intercept
        arcpy.AddMessage(IMessage)
        arcpy.AddMessage('\n')

      #Now onto actually processing the features!

    #List the decision factors and their corresponding numbers, so it's
easier to read the attribute table of the final output
    factorlist(Factors)

    #Create an empty list of Near Fields (predictors)
    #The addnearfield function appends the names of Near Fields to this list
```

```
    #It's a global variable because it is used in the rawscore function as
well
    Nearfields = []

    #LoopcCount corresponds to how many times addnearfield has been run
    #But also for what number factor it's being run for
    LoopCount = 1
    #Loop through the decision factors
    for F in Factors:
        #Add a near field for each decision factor, this is the predictor
variable
        addnearfield(Schools,F,LoopCount)
        #Add to the LoopCount so next time the loop runs,
        # it adds a near field for the next sequential decision factor
        LoopCount = LoopCount + 1

    #If the Geographically Weighted Regression box is unchecked, run OLS
Regression
    if GWRBool == "false":
        #Add a field for all the OLS weights and a field for the intercept
        getweightOLS(Schools,OLSWeightList)
        #Add a message in the processing window to show which
        #beta-coefficient/weight belongs to which predictor
        displayweightlist(OLSWeightList)
    #If the Geographically Weighted Regression box is checked,
    # run Geographically Weighted Regression
    else:
        #Use Geographically Weighted Regression to create weight fields\
        # for the decision factors, as well as the intercept field
        getweightsGWR(Schools,Nearfields,PredictionField)

    #Create a rawscore for the RiskFeatures
    # by entering the created fields into the regression equation
    rawscore(Schools)
    #Rank the Raw Risk Scores from largest to smallest
    rank(Schools,"RawRisk")


#Error Handling
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")
```