Pyomo Code Documentation

1. Purpose and Description of this README File

The purpose of this documentation is to describe the purpose of each major segment of code (henceforth referred to as *code block*) in the Python jupyter Notebook file: *MultiObjectiveModel_Joplin_Pyomo.ipynb*. The purpose of the referred-to Python code file is to run a series of linear programming models for single- and multi-objective optimization relating to extreme weather's effect on a community in terms of three objective functions. The three objectives used in this program are to minimize economic loss, minimize population dislocation, and maximize building functionality. The data used as input files to this program have been preprocessed in a separate program, they are not raw data files. This program is written in Python using the open-source Pyomo modeling library.

Project Authors and Contributors:

- *Tarun Alduri*
- *Sai Bhavaraju*
- *Dale Cochran – Code Author*
- *Andres Gonzalez*
- *Charles Nicholson*
- *Yunjie Wen*

Contact:

Charles Nicholson: cnicholson@ou.edu

2. Code blocks detail of Jupyter Notebook.

2.1 Code Block 1

The first code block imports the necessary Python libraries and packages for this program. The most prominent library used in this program is *Pyomo*, which is used to define and run the model. The libraries *matplotlib* and *plotly* are used to create visualizations of the results generated from the various runs of the model.

2.2 Code Block 2

The second code block allows the user to set several options for how the remainder of the program will run, such as setting the solver the code will use to optimize the subsequent models or which of the optimization models should be run. If a model is not run, the program will not produce the plots related to the results of that given model. Then the values of the user-defined variables are checked to ensure that the values entered are valid. If the values are not valid, then the program outputs a message related to the specific error and stops the code execution.

Variable in Code	Description
modelSolver	User-defined variable to determine which solver will be used throughout the program to optimize the variations of the model. The program's default is to use the open-source

	CBC solver engine via the NEOS Server solver manager (modelSolver=0). The other two options for this variable are to use Pyomo's open-source GLPK solver engine (modelSolver=1), or to use the Gurobi solver engine (modelSolver=2).
numEpsilonSteps	User-defined variable to determine the number of epsilon values that will be evaluated for each run of the model containing a(n) epsilon constraint(s). This number will be applied to all of the model runs that include one or more epsilon constraints. The program's default is to use 20 epsilon values.
maxBudget	User-defined variable to define the maximum possible budget. This program's default is to define the maxBudget as the monetary amount that would be required to retrofit every building to the best possible strategy level. Acceptable values for this variable are "default" or a numeric value.
availBudget	User-defined variable to define the available budget (the budget amount used to constrain the model) as a percentage of maxBudget. This program's default is to use availBudget as 20% of maxBudget. Acceptable values for this variable are percentages (percentages of maxBudget) entered in decimal form.
runModel1	User-defined variable to determine whether or not to run the model for optimizing economic loss subject to population dislocation epsilon constraints. Acceptable values for this variable are 1 (run the model) or 0 (don't run the model).
runModel2	User-defined variable to determine whether or not to run the model for optimizing economic loss subject to building functionality epsilon constraints. Acceptable values for this variable are 1 (run the model) or 0 (don't run the model).
runModel3	User-defined variable to determine whether or not to run the model optimizing population dislocation subject to economic loss epsilon constraints. Acceptable values for this variable are 1 (run the model) or 0 (don't run the model).
runModel4	User-defined variable to determine whether or not to run the model optimizing population dislocation subject to building functionality epsilon constraints. Acceptable values for this

	variable are 1 (run the model) or 0 (don't run the model).
runModel5	User-defined variable to determine whether or not to run the model optimizing building functionality subject to economic loss epsilon constraints. Acceptable values for this variable are 1 (run the model) or 0 (don't run the model).
runModel6	User-defined variable to determine whether or not to run the model optimizing building functionality subject to population dislocation epsilon constraints. Acceptable values for this variable are 1 (run the model) or 0 (don't run the model).
runModel7	User-defined variable to determine whether or not to run the model optimizing economic loss subject to population dislocation epsilon constraints. Acceptable values for this variable are 1 (run the model) or 0 (don't run the model).
runModel8	User-defined variable to determine whether or not to run the model optimizing population dislocation subject to economic loss epsilon constraints. Acceptable values for this variable are 1 (run the model) or 0 (don't run the model).
runModel9	User-defined variable to determine whether or not to run the model optimizing building functionality subject to economic loss epsilon constraints. Acceptable values for this variable are 1 (run the model) or 0 (don't run the model).
scaleData	User-defined variable to determine whether or not the monetary data *I* (direct economic loss) and *Sc* (strategy level cost) should be scaled by the *scaleQuantity* value when the data is read-in.
scaleQuantity	User-defined variable to determine how the monetary data *I* (direct economic loss) and *Sc* (strategy level cost) should be scaled when the data is read-in.

2.3 Code Block 3

The third code block loads the two required data files from the current working directory. The respective files are loaded from Excel CSV files and stored in the program as Pandas data frames. The first data file, *Data_Q_t_6_basedonage_WS60.csv*, contains data pertaining to economic loss, population dislocation, and building functionality. The second data file, *Data_Sc_basedonage_WS60.csv*, contains data pertaining to the cost of retrofitting buildings from one strategy level to another. Subsequently, two *for loops* are used to ensure the correct

data type for the *Z* data and to conduct any scaling (if desired). The required data and naming conventions for the two data files are provided below:

**Note: The two data files must have the required information below. If the names are different, the files to load into the program must be changed on lines 57 and 63 in the code (in the second code block).*

First Data File	
Required Data	**Required CSV File Column Name**
Block ID	Z
Arch Type	S
Building Strategy Level	K
Expected direct economic loss in block i of structure type j at strategy level k (l_{ijk}).	l
Quantity of building prior to any retrofit actions in block i of structure type j at strategy level k (b_{ijk}).	b
Parameter required for model.	r_ijk
Functionality of building in block i of structure type j at strategy level k at time t ($\widehat{Q_{ijk}}$).	Q_t_hat
Population dislocation in building in block i of structure type j at strategy level k.	d_ijk

Second Data File	
Required Data	**Required CSV File Column Name**
Block ID	Z
Arch Type	S
Building Strategy Level	K
Parameter required for model.	r_ijk
Building strategy level after retrofit actions.	K'
The cost of retrofitting a building in block i of structure type j from initial strategy level k to strategy level k'.	Sc

Variable in Code	**Description**
myData	Pandas data frame to hold the information loaded from the first data file.
myData_Sc	Pandas data frame to hold the information loaded from the second data file.

2.4 Code Block 4

The first major section of the fourth code block creates and defines the base model for the remainder of the program. Initially the model is declared as a Pyomo *concrete model* and then the required indices and sets that will be used throughout the model are declared and initialized. Next the decision variables, parameters, and their respective domains are declared, and data for

said parameters is initialized from the *myData* and *myData_Sc* data frames (the data frames that contain the required information loaded in the previous code block). The total (max) possible budget is declared as a parameter and is the budget required to retrofit every building to the best possible strategy level. Subsequently, the available budget for the model is initialized as 20% of the total (max) possible budget. Next the three objective functions of interest are defined as Python functions, followed by the base constraints for the model (the constraints prior to considering any epsilon constraints). The objective of *minimizing economic loss* is defined in the Python function *obj_economic* and is declared as the Pyomo objective *model.objective_1*. The objective of *minimizing population dislocation* is defined in the Python function *obj_dislocation* and is declared as the Pyomo objective *model.objective_2*. The objective of *maximizing building functionality* is defined in the Python function *obj_functionality* and is declared as the Pyomo objective *model.objective_3*. Finally, this section of the code block declares the solver manager for the program on code line 316 as the NEOS server (the CBC engine will be declared later on).

Variable in Code	**Description**
model	The Pyomo concrete model.
model.Z	Set of all unique block ID numbers in the *Z* column of the *myData* data frame.
model.S	Set of all unique archtypes in the *S* column of the *myData* data frame.
model.K	Set of all unique numbers in the *K* column of the *myData* data frame.
model.K_prime	Set of all unique numbers in the *K'* column of the *myData_Sc* data frame.
zsk	List of all the block ID, archtype, and strategy level combinations in the *myData* data frame.
model.ZSK	Pyomo model set of all block ID, archtype, and strategy level combinations in the *myData* data frame.
zs	List of all the block ID and archtype combinations in the *myData* data frame.
model.ZS	Pyomo model set of all block ID and archtype combinations in the *myData* data frame.
kk_prime	List of all possible strategy level retrofit actions.
model.KK_prime	Pyomo model set of all possible strategy level retrofit actions from initial level K to final level K_prime.
k_primek	List of all combinations of final strategy level k_prime and initial strategy level k.
model.K_primeK	Pyomo model set of all combinations of final strategy level K_prime and initial strategy level K.
zskk_prime	List of all combinations of block ID, archtype, initial strategy level, and final strategy level from the *myData_Sc* data frame.
model.ZSKK_prime	Pyomo model set of all combinations of block ID, archtype, initial strategy level, and final strategy level from the *myData_Sc* data frame.

model.x_ijk	Pyomo decision variable for the total number of buildings in neighborhood i of structure type j at strategy level k after retrofitting.
model.y_ijkk_prime	Pyomo decision variable for the total number of buildings in neighborhood i of structure type j retrofitted from strategy level k to strategy level k_prime.
model.l_ijk	Pyomo economic loss cost parameter for buildings in neighborhood i of structure type j at strategy level k.
model.d_ijk	Pyomo population dislocation parameter for buildings in neighborhood i of structure type j at strategy level k.
model.b_ijk	Pyomo parameter for total number of buildings in neighborhood i of structure type j at strategy level k.
model.Q_t_hat	Pyomo Building functionality parameter for buildings in neighborhood i of structure type j at strategy level k.
model.Sc_ijkk_prime	Pyomo parameter for the cost of retrofit actions to retrofit a building in neighborhood i of structure type j from strategy level k to strategy level k_prime.
sumSc	The total cost required to retrofit all buildings in all neighborhoods $i \in Z$ of all structure types $j \in S$ to the best possible strategy level.
model.B	Pyomo parameter for the available budget for retrofit actions (20% of sumSc).
obj_economic	Function for defining the objective of minimizing the direct economic loss (first objective function).
model.objective_1	Pyomo declaration of direct economic loss objective function.
model.econ_loss	Pyomo parameter to hold and monitor the direct economic loss.
obj_dislocation	Function for defining the objective of minimizing the population dislocation (second objective function).
model.objective_2	Pyomo declaration of population dislocation objective function.
model.dislocation	Pyomo parameter to hold and monitor the population dislocation.
obj_functionality	Function for defining the objective of maximizing building functionality (third objective function).
model.objective_3	Pyomo declaration of building functionality objective function.
model.functionality	Pyomo parameter to hold and monitor the building functionality.

retrofit_cost_rule	Function for defining the constraint that the total cost of all retrofit actions taken must be less than or equal to the available budget (*model.B*).
model.retrofit_budget_constraint	Pyomo declaration of the *retrofiti_cost_rule* model constraint.
number_buildings_ij_rule	Function for defining the constraint that the total number of buildings in neighborhood i of structure type j must remain the same before and after retrofit actions (e.g., building strategy levels can change, but buildings themselves cannot be built, destroyed, or moved).
model.number_buildings_ij_constraint	Pyomo declaration of the *number_buildings_ij_rule* model constraint.
model.a	Pyomo parameter to hold the value of y_ijk_primek for the *building_level_rule* model constraint.
model.c	Pyomo parameter to hold the value of y_ijkk_prime for the *building_level_rule* model constraint.
building_level_rule	Function for defining the constraint that the number of buildings retrofitted from initial strategy level k to final strategy level k_prime must equal the number of buildings that were retrofitted to final strategy level k_prime from initial strategy level k.
model.building_level_constraint	Pyomo declaration of the *building_level_rule* model constraint.
solver_manager	Declaration of the Pyomo solver manager to use for optimizing the model (NEOS server).

The second major section in this code block solves the base model (defined in the previous major section of this code block) for each of the three objective functions respectively, without any epsilon constraints. If the solver returns an optimal solution, the respective results from each solve are saved into a dataframe (respective to each of the three runs of the model) for later use to determine the feasible range of epsilon values. For each solve (three total in this code block) the objective function being evaluated is first activated and the other two are deactivated; the activated objection function is what the Pyomo model will optimize while the other deactivated objective functions are ignored. For example, if *model.objective_1* is activated and *model.objective_2* and *model.objective_3* are deactivated, then when the model is solved it will optimize (minimize) total economic loss without regard to the population dislocation or the building functionality. After the program solves the model for each of the three objective functions separately (i.e., without any epsilon constraints), the resulting economic loss, population dislocation, and building functionality from each of the separate solves is output, as well as the maximum possible budget (*sumSc* variable) and the total available budget used for optimization (*model.B*, 20% of *sumSc*).

Variable in Code	Description
model.objective_1.activate()	Pyomo call to activate the first objective function (minimize total economic loss). *Note: This call can be applied to any objective function (e.g., model.objective_2 or model.objective_3).*
model.objective_2.deactivate()	Pyomo call to deactivate the second objective function (minimize population dislocation). *Note: This call can be applied to any objective function (e.g., model.objective_1 or model.objective_3).*
results	Python variable identifier to hold the results of the optimization performed on the given model. The solver option *opt* is set to *cbc* to declare that the CBC solver engine (on the NEOS server) will be used to solve and optimize the model.
obj_1_min_epsilon	Python variable to hold the minimum direct economic loss resulting from optimizing *model.objective_1*. This value corresponds to the minimum epsilon value in the epsilon feasible range for direct economic loss when later conducting multi-objective optimization.
obj_2_value_1	Python variable to hold the population dislocation that resulted from optimizing (minimizing) direct economic loss by itself.
obj_3_value_1	Python variable to hold the building functionality that resulted from optimizing (minimizing) direct economic loss by itself.
obj_2_min_epsilon	Python variable to hold the minimum population dislocation resulting from optimizing *model.objective_2*. This value corresponds to the minimum epsilon value in the epsilon feasible range for population dislocation when later conducting multi-objective optimization.
obj_1_value_2	Python variable to hold the direct economic loss that resulted from optimizing (minimizing) population dislocation by itself.
obj_3_value_2	Python variable to hold the building functionality that resulted from optimizing (minimizing) population dislocation by itself.
obj_3_max_epsilon	Python variable to hold the maximum building functionality resulting from optimizing *model.objective_3*. This value corresponds to the maximum epsilon value in the epsilon feasible range for building functionality when later conducting multi-objective optimization.

obj_1_value_3	Python variable to hold the direct economic loss that resulted from optimizing (maximizing) building functionality by itself.
obj_2_value_3	Python variable to hold the population dislocation that resulted from optimizing (maximizing) building functionality by itself.

The third major section in this code block declares, initializes, and displays the feasible ranges for the direct economic loss epsilon values, population dislocation epsilon values, and building functionality epsilon values as Pyomo parameters of the model. Subsequently, this code block declares and initializes the "step size" for each epsilon range necessary to evaluate 21 respective epsilon values when conducting multi-objective optimization with epsilon constraints. The minimum economic loss epsilon value is obtained from optimizing objective function 1 without any epsilon constraints.

The maximum economic loss epsilon value is the largest economic loss value that resulted between when objective function 2 (dislocation) was solved without any epsilon constraints and when objective function 3 (functionality) was solved without any epsilon constraints. The minimum dislocation epsilon value is obtained from optimizing objective function 2 without any epsilon constraints. The maximum dislocation epsilon value is the largest dislocation value that resulted between when objective function 1 (economic loss) was solved without any epsilon constraints and when objective function 3 (functionality) was solved without any epsilon constraints. The maximum functionality epsilon value is obtained from optimizing objective function 3 without any epsilon constraints. The minimum functionality epsilon value is the smallest functionality value that resulted between when objective function 1 (economic loss) was solved without any epsilon constraints and when objective function 2 (dislocation) was solved without any epsilon constraints.

Variable in Code	**Description**
model.econ_loss_max	Pyomo parameter for the maximum direct economic loss epsilon value.
model.econ_loss_min	Pyomo parameter for the minimum direct economic loss epsilon value (result of optimizing direct economic loss by itself).
model.dislocation_max	Pyomo parameter for the maximum population dislocation epsilon value.
model.dislocation_min	Pyomo parameter for the minimum population dislocation epsilon value (result of optimizing population dislocation by itself).
model.functionality_max	Pyomo parameter for the maximum building functionality epsilon value (result of optimizing building functionality by itself).
model.functionality_min	Pyomo parameter for the minimum building functionality epsilon value.
model.econ_loss_step	Pyomo parameter for the "step size" necessary to evaluate 21 direct economic loss epsilon values within the respective feasible

	epsilon range (between *model.econ_loss_min* and *model.econ_loss_max*).
model.dislocation_step	Pyomo parameter for the "step size" necessary to evaluate 21 population dislocation epsilon values within the respective feasible epsilon range (between *model.dislocation_min* and *model.dislocation_max*).
model.functionality_step	Pyomo parameter for the "step size" necessary to evaluate 21 building functionality epsilon values within the respective feasible epsilon range (between *model.functionality_min* and *model.functionality_max*).

2.5 Code Block 5

The fifth code block runs several variants of the base model to conduct multi-objective optimization. The first six runs of the model in this code block optimize each combination of optimizing one objective function with one of the other objectives set as an epsilon constraint. The last three runs of the model in this code block optimize each of the three objective functions with the other two objectives set as epsilon constraints.

The first model run optimizes the first objective function (minimize direct economic loss) with the base model constraints in addition to an epsilon constraint for the second objective (minimize population dislocation). The model is run, optimized, and the results are recorded/displayed 21 times for the 21 distinct population dislocation epsilon values. For each run of the model in this code block, the respective epsilon constraint is added to the model prior to optimization and then subsequently removed from the model after optimization. There are 21 epsilon values/constraints evaluated because there are 20 "steps" between the minimum and maximum population dislocation epsilon values (including the minimum, but not including the maximum, so the 21st epsilon value evaluated is the maximum in the epsilon range).

Variable in Code	**Description**
obj_1_2_epsilon_results	Pandas data frame to hold the population dislocation epsilon value used in the epsilon constraint and the resulting direct economic loss (objective), population dislocation, and building functionality from each run of the model.
model.obj_2_e	Pyomo parameter to hold the population dislocation epsilon value for the epsilon constraint in the respective run of the model. The epsilon constraint is declared such that the resulting population dislocation from optimizing the model must be less than or equal to the given epsilon value.

This second model run optimizes the first objective function (minimize direct economic loss) with the base model constraints in addition to an epsilon constraint for the third objective (maximize building functionality). The model is run, optimized, and the results are recorded/displayed 21 times for the 21 distinct building functionality epsilon values. For each run of the model in this code block, the respective epsilon constraint is added to the model prior to optimization and then subsequently removed from the model after optimization. There are 21 epsilon values/constraints evaluated because there are 20 "steps" between the minimum and maximum building functionality epsilon values (including the minimum, but not including the maximum, so the 21st epsilon value evaluated is the maximum in the epsilon range).

Variable in Code	Description
obj_1_3_epsilon_results	Pandas data frame to hold the building functionality epsilon value used in the epsilon constraint and the resulting direct economic loss (objective), population dislocation, and building functionality from each run of the model.
model.obj_3_e	Pyomo parameter to hold the building functionality epsilon value for the epsilon constraint in the respective run of the model. The epsilon constraint is declared such that the resulting building functionality from optimizing the model must be greater than or equal to the given epsilon value.

The third model run optimizes the second objective function (minimize population dislocation) with the base model constraints in addition to an epsilon constraint for the first objective (minimize direct economic loss). The model is run, optimized, and the results are recorded/displayed 21 times for the 21 distinct direct economic loss epsilon values. For each run of the model in this code block, the respective epsilon constraint is added to the model prior to optimization and then subsequently removed from the model after optimization. There are 21 epsilon values/constraints evaluated because there are 20 "steps" between the minimum and maximum direct economic loss epsilon values (including the minimum, but not including the maximum, so the 21st epsilon value evaluated is the maximum in the epsilon range).

Variable in Code	Description
obj_2_1_epsilon_results	Pandas data frame to hold the direct economic loss epsilon value used in the epsilon constraint and the resulting direct economic loss, population dislocation (objective), and building functionality from each run of the model.
model.obj_1_e	Pyomo parameter to hold the direct economic loss epsilon value for the epsilon constraint in the respective run of the model.

	The epsilon constraint is declared such that the resulting direct economic loss from optimizing the model must be less than or equal to the given epsilon value.

The fourth model run optimizes the second objective function (minimize population dislocation) with the base model constraints in addition to an epsilon constraint for the third objective (maximize building functionality). The model is run, optimized, and the results are recorded/displayed 21 times for the 21 distinct building functionality epsilon values. For each run of the model in this code block, the respective epsilon constraint is added to the model prior to optimization and then subsequently removed from the model after optimization. There are 21 epsilon values/constraints evaluated because there are 20 "steps" between the minimum and maximum building functionality epsilon values (including the minimum, but not including the maximum, so the 21st epsilon value evaluated is the maximum in the epsilon range).

Variable in Code	**Description**
obj_2_3_epsilon_results	Pandas data frame to hold the building functionality epsilon value used in the epsilon constraint and the resulting direct economic loss, population dislocation (objective), and building functionality from each run of the model.
model.obj_3_e	Pyomo parameter to hold the building functionality epsilon value for the epsilon constraint in the respective run of the model. The epsilon constraint is declared such that the resulting building functionality from optimizing the model must be greater than or equal to the given epsilon value.

The fifth model run optimizes the third objective function (maximize building functionality) with the base model constraints in addition to an epsilon constraint for the first objective (minimize direct economic loss). The model is run, optimized, and the results are recorded/displayed 21 times for the 21 distinct direct economic loss epsilon values. For each run of the model in this code block, the respective epsilon constraint is added to the model prior to optimization and then subsequently removed from the model after optimization. There are 21 epsilon values/constraints evaluated because there are 20 "steps" between the minimum and maximum direct economic loss epsilon values (including the minimum, but not including the maximum, so the 21st epsilon value evaluated is the maximum in the epsilon range).

Variable in Code	**Description**
obj_3_1_epsilon_results	Pandas data frame to hold the direct economic loss epsilon value used in the epsilon constraint and the resulting direct economic

	loss, population dislocation, and building functionality (objective) from each run of the model.
model.obj_1_e	Pyomo parameter to hold the direct economic loss epsilon value for the epsilon constraint in the respective run of the model. The epsilon constraint is declared such that the resulting direct economic loss from optimizing the model must be less than or equal to the given epsilon value.

The sixth model run optimizes the third objective function (maximize building functionality) with the base model constraints in addition to an epsilon constraint for the second objective (minimize population dislocation). The model is run, optimized, and the results are recorded/displayed 21 times for the 21 distinct population dislocation epsilon values. For each run of the model in this code block, the respective epsilon constraint is added to the model prior to optimization and then subsequently removed from the model after optimization. There are 21 epsilon values/constraints evaluated because there are 20 "steps" between the minimum and maximum population dislocation epsilon values (including the minimum, but not including the maximum, so the 21st epsilon value evaluated is the maximum in the epsilon range).

Variable in Code	**Description**
obj_3_2_epsilon_results	Pandas data frame to hold the population dislocation epsilon value used in the epsilon constraint and the resulting direct economic loss, population dislocation, and building functionality (objective) from each run of the model.
model.obj_2_e	Pyomo parameter to hold the population dislocation epsilon value for the epsilon constraint in the respective run of the model. The epsilon constraint is declared such that the resulting population dislocation from optimizing the model must be less than or equal to the given epsilon value.

The seventh model run optimizes the first objective function (minimize direct economic loss) with the base model constraints in addition to an epsilon constraint for the second objective (minimize population dislocation) and an epsilon constraint for the third objective (maximize building functionality). The model is run, optimized, and the results are recorded/displayed for each combination of the 21 distinct population dislocation epsilon values and 21 distinct building functionality epsilon values. For this code block, a distinct population dislocation epsilon constraint is added to the model, and then the model is run/optimized along with each distinct building functionality epsilon constraint. Each distinct building functionality epsilon constraint is removed

from the model after optimization and prior to the subsequent optimization with the next building functionality epsilon constraint. This process is then repeated for each of the remaining distinct population dislocation epsilon constraints.

Variable in Code	Description
obj_1_23_epsilon_results	Pandas data frame to hold the population dislocation and building functionality epsilon values used in the epsilon constraints and the resulting direct economic loss (objective), population dislocation, building functionality, and percent of the available budget used for retrofit actions from each run of the model.
model.obj_2_e	Pyomo parameter to hold the population dislocation epsilon value for the epsilon constraint in the respective run of the model. The epsilon constraint is declared such that the resulting population dislocation from optimizing the model must be less than or equal to the given epsilon value.
model.obj_3_e	Pyomo parameter to hold the building functionality epsilon value for the epsilon constraint in the respective run of the model. The epsilon constraint is declared such that the resulting building functionality from optimizing the model must be greater than or equal to the given epsilon value.
budget_used	Python variable to hold the monetary cost of all retrofit actions taken as a result of optimizing the model with the given population dislocation and building functionality epsilon constraints.
percent_budget_used	The percent of the available budget used to conduct the retrofit actions taken as a result of optimizing the model with the given population dislocation and building functionality epsilon constraints.
results_df	Pandas data frame to hold the results of each model run and write them to an Excel CSV file in the working directory.
obj_1_23_data	Pandas data frame to hold the high-level results of the model runs after any infeasible iterations have been dropped. Infeasible iterations may occur (particularly at epsilon value extremes) due to data scaling or timing-out of the solver engine.
obj_1_23_optimal	Pandas data frame to hold the high-level results of the model runs after removing the dominant points

The ninth model run optimizes the second objective function (minimize population dislocation) with the base model constraints in addition to an epsilon constraint for the first objective (minimize direct economic loss) and an epsilon constraint for the third objective (maximize building functionality). The model is run, optimized, and the results are recorded/displayed for each combination of the 21 distinct direct economic loss epsilon values and 21 distinct building functionality epsilon values. For this code block, a distinct direct economic loss epsilon constraint is added to the model, and then the model is run/optimized along with each distinct building functionality epsilon constraint. Each distinct building functionality epsilon constraint is removed from the model after optimization and prior to the subsequent optimization with the next building functionality epsilon constraint. This process is then repeated for each of the remaining distinct direct economic loss epsilon constraints.

Variable in Code	**Description**
obj_2_13_epsilon_results	Pandas data frame to hold the direct economic loss and building functionality epsilon values used in the epsilon constraints and the resulting direct economic loss, population dislocation (objective), building functionality, and percent of the available budget used for retrofit actions from each run of the model.
model.obj_1_e	Pyomo parameter to hold the direct economic loss epsilon value for the epsilon constraint in the respective run of the model. The epsilon constraint is declared such that the resulting direct economic loss from optimizing the model must be less than or equal to the given epsilon value.
model.obj_3_e	Pyomo parameter to hold the building functionality epsilon value for the epsilon constraint in the respective run of the model. The epsilon constraint is declared such that the resulting building functionality from optimizing the model must be greater than or equal to the given epsilon value.
budget_used	Python variable to hold the monetary cost of all retrofit actions taken as a result of optimizing the model with the given direct economic loss and building functionality epsilon constraints.
percent_budget_used	The percent of the available budget used to conduct the retrofit actions taken as a result of optimizing the model with the given direct economic loss and building functionality epsilon constraints.
results_df	Pandas data frame to hold the results of each model run and write them to an Excel CSV file in the working directory.
obj_2_13_data	Pandas data frame to hold the high-level results of the model runs after any infeasible iterations have been dropped. Infeasible

	iterations may occur (particularly at epsilon value extremes) due to data scaling or timing-out of the solver engine.
obj_2_13_optimal	Pandas data frame to hold the high-level results of the model runs after removing the dominant points

The eighth model run optimizes the third objective function (maximize building functionality) with the base model constraints in addition to an epsilon constraint for the first objective (minimize direct economic loss) and an epsilon constraint for the second objective (minimize population dislocation). The model is run, optimized, and the results are recorded/displayed for each combination of the 21 distinct direct economic loss epsilon values and 21 distinct population dislocation epsilon values. For this code block, a distinct direct economic loss epsilon constraint is added to the model, and then the model is run/optimized along with each distinct population dislocation epsilon constraint. Each distinct population dislocation epsilon constraint is removed from the model after optimization and prior to the subsequent optimization with the next population dislocation epsilon constraint. This process is then repeated for each of the remaining distinct direct economic loss epsilon constraints.

Variable in Code	**Description**
obj_3_12_epsilon_results	Pandas data frame to hold the direct economic loss and population dislocation epsilon values used in the epsilon constraints and the resulting direct economic loss, population dislocation, building functionality (objective), and percent of the available budget used for retrofit actions from each run of the model.
model.obj_1_e	Pyomo parameter to hold the direct economic loss epsilon value for the epsilon constraint in the respective run of the model. The epsilon constraint is declared such that the resulting direct economic loss from optimizing the model must be less than or equal to the given epsilon value.
model.obj_2_e	Pyomo parameter to hold the population dislocation epsilon value for the epsilon constraint in the respective run of the model. The epsilon constraint is declared such that the resulting population dislocation from optimizing the model must be less than or equal to the given epsilon value.
budget_used	Python variable to hold the monetary cost of all retrofit actions taken as a result of optimizing the model with the given direct economic loss and population dislocation epsilon constraints.

percent_budget_used	The percent of the available budget used to conduct the retrofit actions taken as a result of optimizing the model with the given direct economic loss and population dislocation epsilon constraints.
results_df	Pandas data frame to hold the results of each model run and write them to an Excel CSV file in the working directory.
obj_3_12_data	Pandas data frame to hold the high-level results of the model runs after any infeasible iterations have been dropped. Infeasible iterations may occur (particularly at epsilon value extremes) due to data scaling or timing-out of the solver engine.
obj_3_12_optimal	Pandas data frame to hold the high-level results of the model runs after removing the dominant points

The eighth code block (code lines 577-585) displays a plot of the results from the seventh code block (optimizing direct economic loss with population dislocations set as an epsilon constraint). In the plot, the direct economic loss (the objective function) is displayed on the y-axis and the population dislocation that resulted from each corresponding run of the model is displayed on the x-axis.

2.6 Code Block 6

The fifth code block in this program creates plots to visualize the results generated from each of the nine multi-objective optimization model runs in the previous code block. Most of the plots generated use the *matplotlib* library and are scatterplots or contour plots. For the model runs that used two epsilon constraints, interactive 3D plots were also generated using the *plotly* library.

3. Run the code in complier

To run this program please follow these argument guidelines:

```
$> python MultiObjectiveModel_Joplin_Pyomo.py -q <Q_Data.csv>
                                               -c <Sc_Data.csv>
                                               [-s <model_solver>
                                               -e <epsilon_steps>
                                               -m <max_budget>
                                               -a <avail_budget>
                                               -r <[run_models]>
                                               -d <scale_data>
                                               -f <scale_quantity>
                                               -v <verbose>]
```

Notice that the input data files are the only hard requirements to run the script. The arguments can be presented as single-character flags or full argument flags as described by the following table:

REQUIRED	Short Flags	**Full Flags**	Description
YES	-q	--Q=	Path to the Q_t data file.
YES	-c	--Sc=	Path to the S_tcdata file.
NO	-s	--solver=	Solver selection (**default: 0**): {0=Neos Server | 1=GLPK | 2=Gurobi}
NO	-e	-epsilon=	Number of epsilon steps (**default: 20**)
NO	-m	--maxBuddget=	Maximum budget amount (**default: "default"**): {**integer value** (usually millions) | "default"=maximum possible budge as the monetary amount that would be required to retrofit every building to the best possible strategy level}
NO	-a	--availBudget	Fraction of the budget that is available to the model(default: 0.2, 20% budget): { decimal value (usually between 0 and 1)}
NO	-r	--runModels=	List of (which of the 9) models to run (**default: 111111111**, run all): {**sequence of 9 binary values**}
NO	-d	--scaleData	Should the input data be scaled? (**default**: 1, scale the data):{1|0}
NO	-f	--scaleQuantity=	Factor by which the data should be scaled (**default: 1000000**, scale data by a million): {**integer value**(usually millions)}
NO	-v	--verbose	Should all output be printed to console?