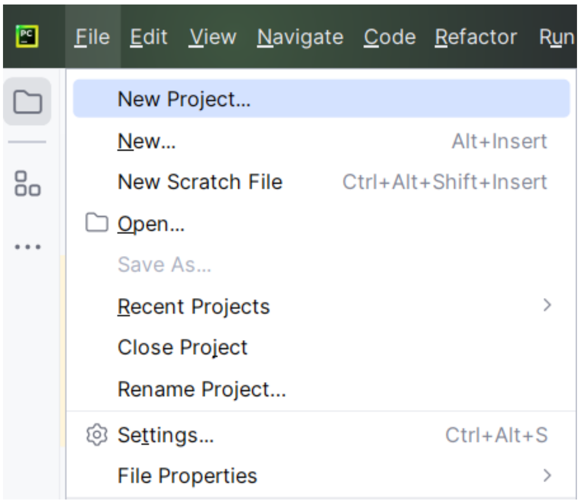
Tutorial for geogxboost python library

This tutorial was created using PyCharm 2023.3.3 (Community Edition).

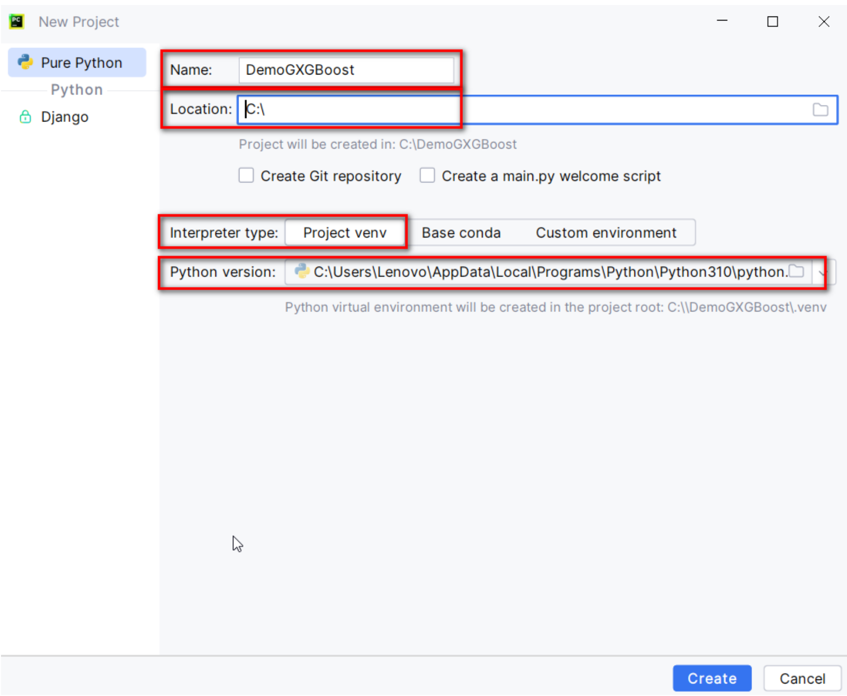
Create project

Open PyCharm and create a new project

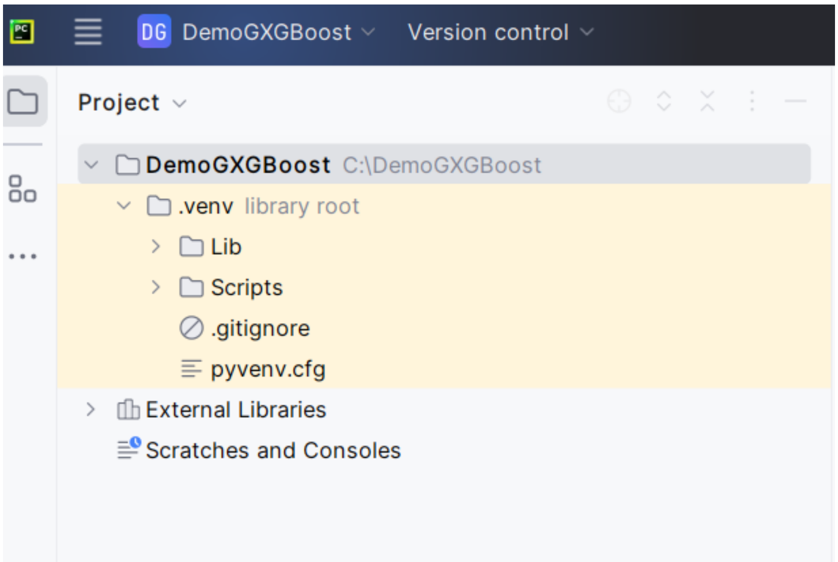
- File->New Project



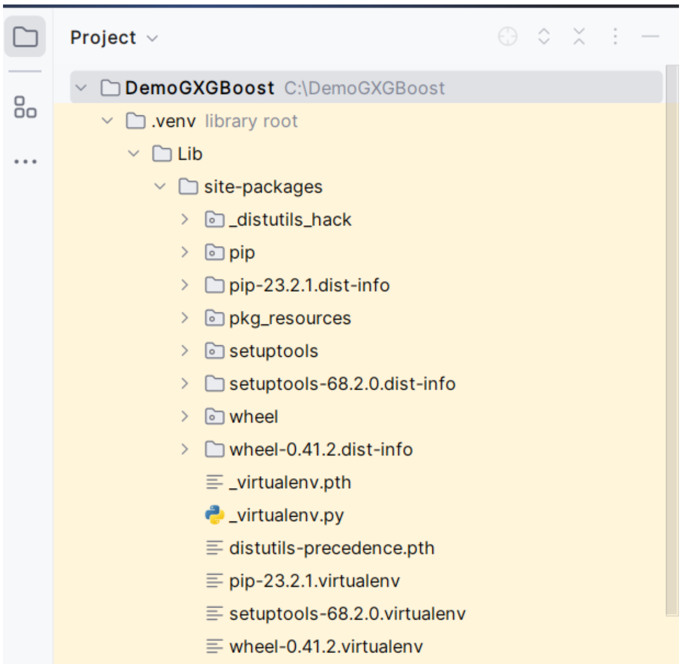
- Set the name of the project and the location as in the image below
- Keep the **Interpreter type** to **Project venv**
- The system will select the Python version that has already been installed. If there is no Python installation, PyCharm will automatically install the latest version
- Click **Create**



The new project and virtual environment are created.



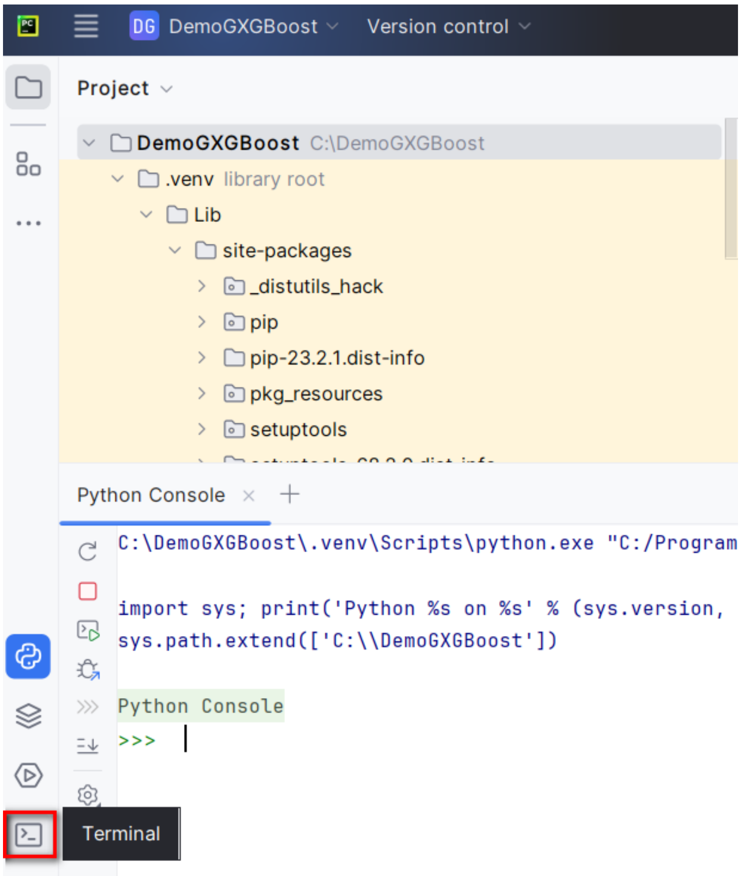
Currently, the `.venv` has only the basic libraries installed (within Lib/site-packages).



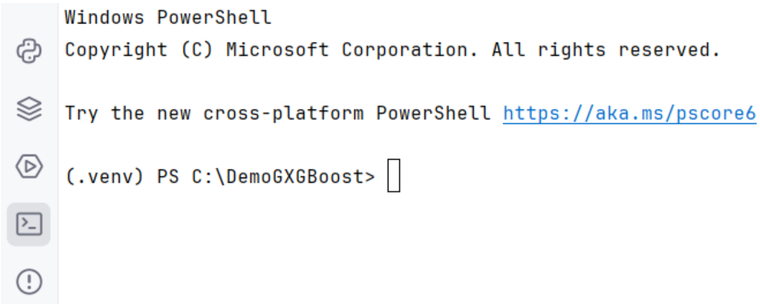
Install geoxgboost library and dependencies

To install `geoxgboost` library, all dependencies should be installed because the library is currently in test mode.

- Click the button to switch from Console to Terminal



- Terminal opens



It shows the current `.venv` and the location `C:\DemoGXGBoost` where `geoxgboost` and its dependencies will be installed.

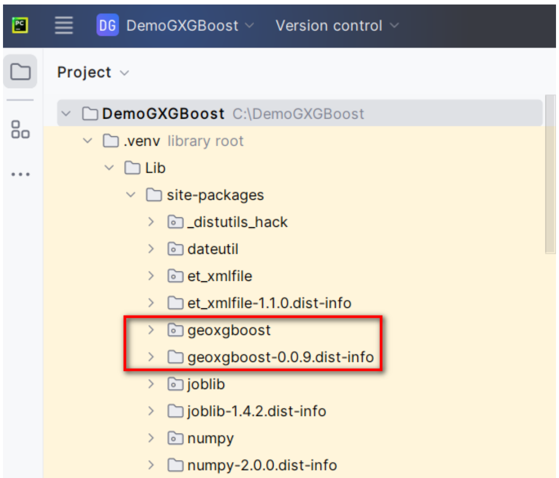
On terminal type:

```
(.venv) PS C:\DemoGXGBoost> pip install numpy
(.venv) PS C:\DemoGXGBoost> pip install pandas
(.venv) PS C:\DemoGXGBoost> pip install scikit-learn
(.venv) PS C:\DemoGXGBoost> pip install scipy
(.venv) PS C:\DemoGXGBoost> pip install xgboost
(.venv) PS C:\DemoGXGBoost> pip install openpyxl
```

To install geoxgboost type:

```
(.venv) PS C:\DemoGXGBoost> pip install -i https://test.pypi.org/simple/
geoxgboost==0.1.0
```

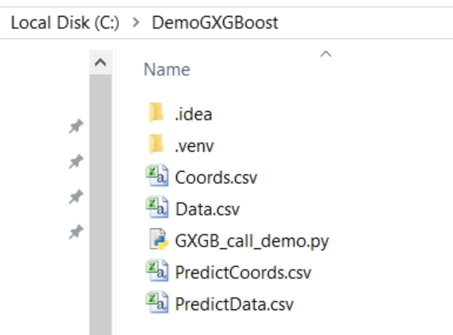
`geoxgboost` is installed within the site-packages folder



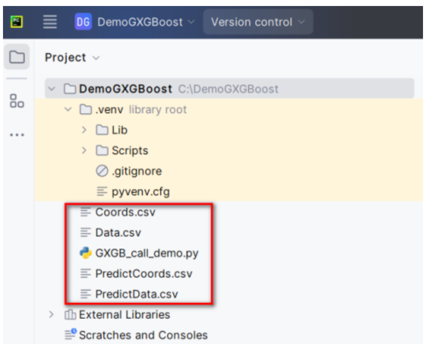
Download demo data

- Download data from https://github.com/georeko/geoxgboost/tree/main/DemoData and save them within the folder `C:\DemoGXGBoost`
- Download python test code `GXGB_call_demo.py` https://github.com/georeko/geoxgboost/tree/main/DemoData and save it within the folder `C:\DemoGXGBoost`

The folder should look like this:



The project folder should look like this:



Data refer to the **Boston housing dataset** (Supplementary Note 1 presents more details on the dataset)

Coords.csv: Includes the coordinates of the spatial units

Coords.csv

	A	B
1	Xcoord	Ycoord
2	2029160	801587
3	2028610	805203
4	2029850	805016
5	2030190	806266
6	2030520	806934
7	2030780	807699
8	2029440	806527
9	2028940	807899
10	2029240	808211

Data.csv: dependent and independent variables

Data.csv

	GIS_id	Independent variables													Dependent	
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	TRACT	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	CMEDV	
2	2011	0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24	
3	2021	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6	
4	2022	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	
5	2031	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	
6	2032	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2	
7	2033	0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7	
8	2041	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9	
9	2042	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	22.1	
10	2043	0.21124	12.5	7.87	0	0.524	5.631	100	6.0821	5	311	15.2	386.63	29.93	16.5	

PredictCoords.csv: Contains the coordinates of the spatial units where the prediction will occur.

PredictData.csv: Contains the values of the independent variables referring to the spatial units in which the prediction will take place.

GXGB_call_demo.py: Demo Python file for analyzing the Boston housing dataset.

Run the code

On PyCharm double-click `GXGB_call_demo.py`

Step 1 Import libraries and data

Data.csv contains a `GIS_id` field, independent variables `X`, and the dependent `y`.

For this reason, Data.csv are cleaned to create `DataFrame X`, containing only the independent variables, and `DataFrame y`, containing only the dependent variable.

- Select rows 1-10 and click `Alt+Shift+E` to run the selection.

GXGB_call_demo.py

```
1  ## Step 1 Import libraries and data
2  #Import libraries
3  import geoxgboost as gx   #Imports geoxgboost
4  import pandas as pd
5  # Import data
6  Coords= pd.read_csv('Coords.csv' ) # Coordinates of centroid
7  Data = pd.read_csv('Data.csv')     # Data including GISid, X(
8  X= Data.iloc [:, 1 : -1]           # Remove GISid and y from
9  y= Data.iloc [:, -1]               # Dependent y
10 VarNames = X.columns[:]            # Get variables' names. Us
```

Results are presented in **Python Console Special Variables list** (right window)

Step 2 Hyperparameter tuning.

Run code in Step 2.

```
12 ## Step 2 Hyper parameter tuning. Define initial hyperparameters for inner loop
13 params= {
14     'n_estimators':100,     #default is 100
15     'learning_rate':0.1,    #default is 0.3
16     'max_depth':6,          #default is 6
17     'min_child_weight':1,   #default is 1
18     'gamma':0,              #default is 0
19     'subsample':0.8,        #default is 1
20     'colsample_bytree':0.8, #default is 1
21     'reg_alpha':0,          #default is 0
22      'reg_lambda':1,        #default is 1
23     }
24 # Define search space for hyperparameteres of inner loop. A maximum of 3 hyperparameters can be
25 Param1=None; Param2=None; Param3=None  # Set hyperparamters to None to avoid overlapping if the
26 Param1_Values = []; Param2_Values = []; Param3_Values = []
27 # Set hyperparameters and values according to the problem. Select and deselect for one or more h
28 Param1='n_estimators'
29 Param1_Values = [100, 200, 300, 500]
30 Param2='learning_rate'
31 Param2_Values = [0.1, 0.05,0.01]
32 Param3='max_depth'
33 Param3_Values = [2,3,5,6]
34 #Create grid
35 param_grid= gx.create_param_grid(Param1,Param1_Values,Param2,Param2_Values,Param3,Param3_Values)
```

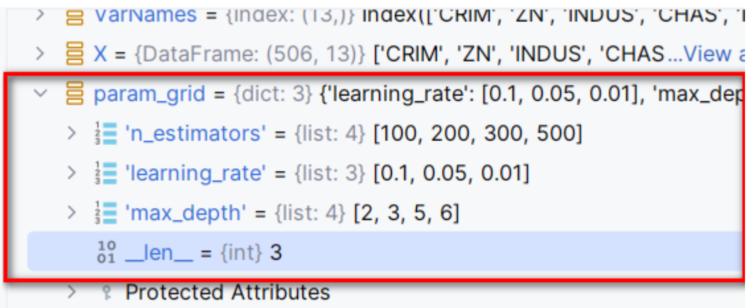
`Params` are the initial hyperparameter values (either user-defined or by default values).

Up to three hyperparameters can be tuned concurrently. To tune only two hyperparameters, comment on `Param3` and `Param3_values` (add the pound sign `#` in front of lines 32 and 33).

`Param1`: first hyperparameter name e.g., `'n_estimators'`
`Param1_Values`: values for search e.g., `[100, 200, 500]`
`Param2`: second hyperparameter name e.g., `'learning_rate'`. Default=None.
`Param2_Values`: values for search e.g., `[0.1, 0.05,0.01]`. Default=None.
`Param3`: third hyperparameter name e.g., `'max_depth'`. Default=None.
`Param3_Values`: values for search e.g., `[2,3,4,6]`. Default=None.

Any tree booster hyperparameter available in XGBoost can be applied here. A complete list of available tree booster hyperparameters can be found at https://xgboost.readthedocs.io/en/stable/parameter.html#parameters-for-tree-booster

The output `param_grid` is presented at the Console Special Variables list.



Step 3 Nested CV to tune hyperparameters

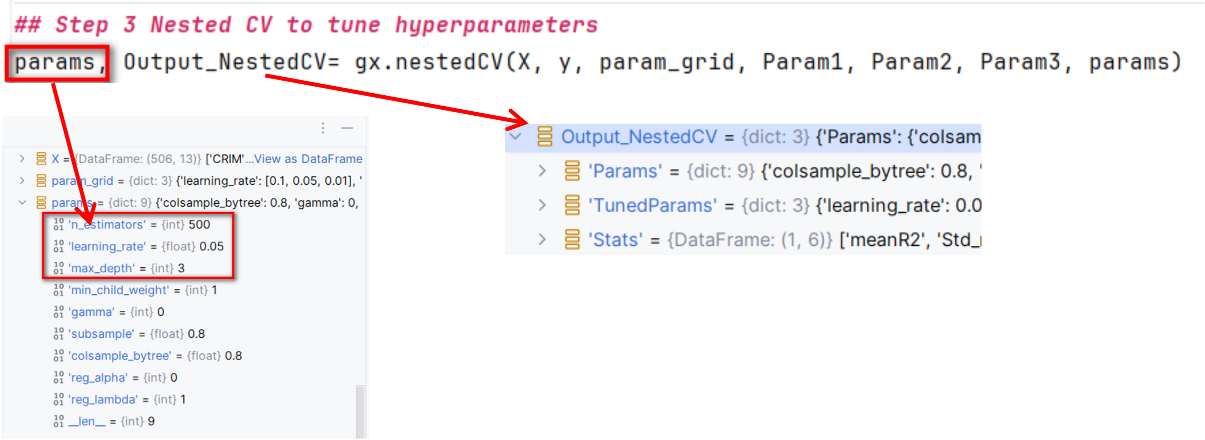
```
## Step 3 Nested CV to tune hyperparameters
params, Output_NestedCV= gx.nestedCV(X, y, param_grid, Param1, Param2, Param3, params)
```

Run step 3 to get the optimized hyperparameters and the model's generalization error.

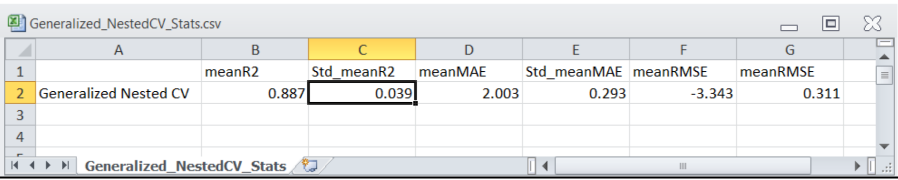
Console output:

```
>>> params, Output_NestedCV= gx.nestedCV(X, y, param_grid, Param1, Param2, Param3, params)
...
=================Nested CV process==========================================
Tuning with 3 hyperparameters
>Count=1,  R2=0.900, RMSE=-3.744, MAE=2.403,cfg={'learning_rate': 0.1, 'max_depth': 2, 'n_estimators':
Tuning with 3 hyperparameters
>Count=2,  R2=0.833, RMSE=-2.973, MAE=2.131,cfg={'learning_rate': 0.05, 'max_depth': 3, 'n_estimators':
Tuning with 3 hyperparameters
>Count=3,  R2=0.928, RMSE=-3.675, MAE=1.612,cfg={'learning_rate': 0.05, 'max_depth': 5, 'n_estimators':
Tuning with 3 hyperparameters
>Count=4,  R2=0.851, RMSE=-3.225, MAE=2.145,cfg={'learning_rate': 0.1, 'max_depth': 3, 'n_estimators':
Tuning with 3 hyperparameters
>Count=5,  R2=0.924, RMSE=-3.097, MAE=1.722,cfg={'learning_rate': 0.1, 'max_depth': 3, 'n_estimators':
=================Nested CV results for hyperparamtre tuning=================
Generalization error: mean-R2 (stdev): 0.887 (0.039)
Mean MAE: 2.003 (0.293)
Mean RMSE: -3.343 (0.311)
Best params taken at model with minimum RMSE at count: 2
```

Metrics for every inner loop and the generalization error of the output loop are presented. The output `params` and `Output_NestedCV` are presented at the Console Special Variables list. `params` include the intial hypermeter values updated with the optimized hyperparameter values. `Output_NestedCV` contains: `Prams values`, `TunedParams` (only the fine-tuned hyperparameters), and `Stats`. `Stats` stores the generalization statistics of the nested cv process: `mean R2`, `mean MAE`, `mean RMSE`, and their standard deviations (`std`). `Stats` is also saved as `xls` file.



The `csv` file containing `Stats` of the nested cv process is saved in `C:\DemoGXGBoost` (unless a different path is specified in the `path_save` property).



Generalized_NestedCV_Stats.csv

	A	B	C	D	E	F	G
1		meanR2	Std_meanR2	meanMAE	Std_meanMAE	meanRMSE	meanRMSE
2	Generalized Nested CV	0.887	0.039	2.003	0.293	-3.343	0.311
3							
4							

Generalized_NestedCV_Stats

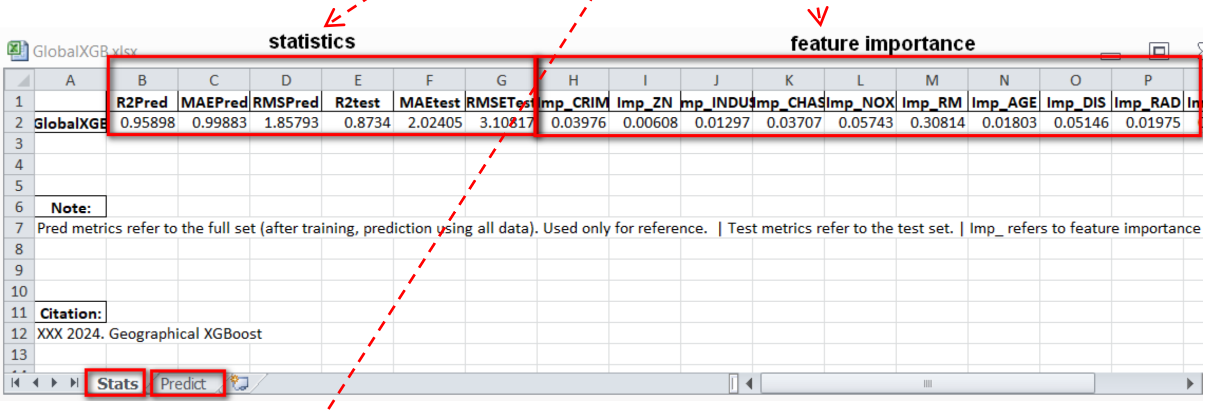
File is exported in `csv` format as `Generalized_NestedCV.csv`

Step 4 GlobalXGBoost model

```python
## Step 4 GlobalXGBoost model
Output_GlobalXGBoost=gx.global_xgb(X,y,params)
```

Console output:

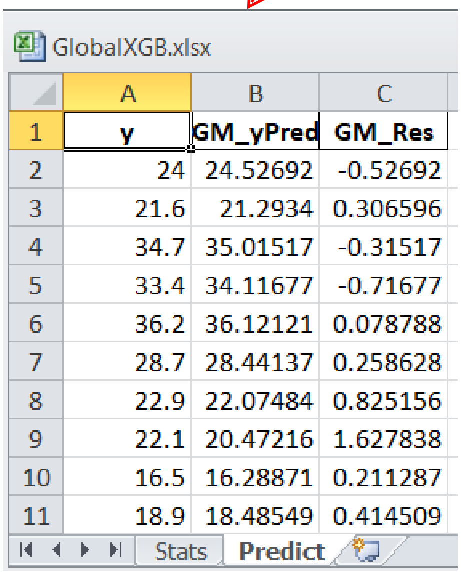
Output: Global feature importance and evaluation metrics in console and xls file.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1		R2Pred	MAEPred	RMSPred	R2test	MAEtest	RMSETest	Imp_CRIM	Imp_ZN	Imp_INDUS	Imp_CHAS	Imp_NOX	Imp_RM	Imp_AGE	Imp_DIS	Imp_RAD
2	GlobalXGB	0.95898	0.99883	1.85793	0.8734	2.02405	3.10817	0.03976	0.00608	0.01297	0.03707	0.05743	0.30814	0.01803	0.05146	0.01975
6	Note:															
7	Pred metrics refer to the full set (after training, prediction using all data). Used only for reference. | Test metrics refer to the test set. | Imp_ refers to feature importance															
11	Citation:															
12	XXX 2024. Geographical XGBoost															

Stats | Predict

`Predict` is in a separate worksheet having: `y` (true value of y), `GM_yPred` (global model y prediction), and `GM_Res` (global model residuals).



	A	B	C
1	y	GM_yPred	GM_Res
2	24	24.52692	-0.52692
3	21.6	21.2934	0.306596
4	34.7	35.01517	-0.31517
5	33.4	34.11677	-0.71677
6	36.2	36.12121	0.078788
7	28.7	28.44137	0.258628
8	22.9	22.07484	0.825156
9	22.1	20.47216	1.627838
10	16.5	16.28871	0.211287
11	18.9	18.48549	0.414509

File is exported in `xlsx` format as `Global_XGB.xlsx`

Step 5 Optimize Bandwidth

```python
## Step 5 Optimize Bandwidth
bw= gx.optimize_bw(X,y, Coords, params, bw_min=75, bw_max=80,step=1, Kernel='Adaptive', spatial_weights=True)
```

Optimal bandwidth can be computationally expensive. Instead of an exhaustive search for an extensive range of values (e.g., 30 to 100 number of nearest neighbors), use a large incremental step (e.g., 5 or 10) and then search around the bandwidth value that minimizes the cross-validation criterion by reducing the step size.

Console output:

```
================Calculating optimal bandwidth========================
Calculation with spatial weights
Adaptive Kernel used
Calculating bw= 75, with bw_max=80 and step of 1
bw= 75, with CV= 3991.947
Calculating bw= 76, with bw_max=80 and step of 1
bw= 76, with CV= 4023.694
Calculating bw= 77, with bw_max=80 and step of 1
bw= 77, with CV= 3958.676
Calculating bw= 78, with bw_max=80 and step of 1
bw= 78, with CV= 3918.717
Calculating bw= 79, with bw_max=80 and step of 1
bw= 79, with CV= 3953.804
Calculating bw= 80, with bw_max=80 and step of 1
bw= 80, with CV= 3950.291
Best bandwidth value: 78 at min CV= 3918.717.
Results have been saved in csv format at the specified path
=====================================================================
```

The function calculates `R2`, `MAE`, `RMSE`, and `CV` (cross-validation criterion). Optimal bandwidth (`BW`) is the value that minimizes `CV`.

BW_results.csv

	A	B	C	D	E	F
1		BW	R2	MAE	RMSE	CV
2	0	75	0.906243	1.96089	2.808776	3991.947
3	1	76	0.905498	1.969347	2.819923	4023.694
4	2	77	0.907025	1.956474	2.797047	3958.676
5	3	78	0.907963	1.949388	2.782894	3918.717
6	4	79	0.907139	1.949232	2.795325	3953.804
7	5	80	0.907222	1.9485	2.794083	3950.291

BW_results

File is exported in `csv` format as `BW_results.csv`

Step 6 GXGB (Geographical-XGBoost)

```
## Step 6 GXGB (Geographical-XGBoost)
Output_GXGB_LocalModel= gx.gxgb(X,y,Coords, params,bw=bw, Kernel='Adaptive', spatial_weights=True, alpha_wt_type='varying', alpha_wt=0.5)
```

The above example uses spatial weights (`spatial_weights=True`) and a varying alpha weight (`alpha_wt=0.5`), leading to an ensemble solution.

```
Output_GXGB_LocalModel = {dict: 6} {'Params': ['Value'] [n_estimators ... View
  'Params' = {DataFrame: (19, 1)} ['Value'] [n_estimators ...View as DataFrame
  'Stats' = {DataFrame: (1, 12)} ['R2_Pred', 'MAE_Pred', 'RMS...View as DataFrame
  'Prediction' = {DataFrame: (506, 25)} ['IDS', 'y', 'LM_yPred...View as DataFrame
  'alpha_wt' = {list: 506} [1, 1, 0.5, 1, 0.5, 0.5, 0.5, 1, 1, 0.5, 1, 0.5, 1, 0.5, '... View
  'y_G_hat' = {list: 506} [np.float32(25.901796), np.float32(20.969429), n... View
  'bestLocalModel' = {list: 506} [XGBRegressor(base_score=None, boost... View
  __len__ = {int} 6
```

GXGB.xlsx

Ensemble model

	R2_Pred	MAE_Pred	RMS_Pred	R2_oob	MAE_oob	RMS_oob	R2oobGl	MAEoobG	RMSEoobGl	R2ens	MAEens	RMSEens
GXGB	0.99998	0.03015	0.04368	0.90796	1.94939	2.78289	0.89121	2.06443	3.025565926	0.93524	1.60133	2.33442

settings and hyperparameters

	Value
n_estimators	500
learning_rate	0.05
max_depth	3
min_child_weight	1
gamma	0
subsample	0.8
colsample_bytree	0.8
reg_alpha	0
reg_lambda	1
Spatial Units	506
Features	13
Kernel	Adaptive
Bandwidth	78
Spatial Weights	TRUE
Alpha Weight type	varying
Alpha Weight value	0.5
Feature Importance	gain
Test Size	0.33
Seed	7

Stats | LocalModels

	Description
IDS	Spatial unit id
y	y_true
LM_yPred	Local model prediction including central point
LM_yOOB	Local model prediction excluding central point
LM_ResOOB	Local model OOB residuals
LMRsqr	Local model Rsqr (oob)
LM_Best_score(RMSE)	Local model best score (oob)
alpha_wt	alpha weight value
yGhat	Prediction of the global model excluding central point
y_ensemble	prediction of y through ensemble
Imp_	Feature local importance
MaxImportance	Maximum feature importance
MaxFeatureID	Feature with max importance

GXGB.xlsx

IDS	y	LM_yPred	LM_yOOB	M_ResOO	LMRsqr	est_score(	alpha_wt	yGhat	_ensembl	Imp_CRIM	Imp_ZN	p_IND	_CH	Imp_NOX	Imp_RM	Imp_AGE	p_	b_	R	b_	TPT	np_	LS	MaxImportance	MaxFeatureID
0	24	24.00051	25.84475	-1.84475	0.712214	-2.79944	1	25.9018	25.84475	0.013757	0	0.01	0	0.011	0.117	0.161	#	#	#	#	#	#	0.195	13
1	21.6	21.55433	21.12187	0.478126	0.673365	-2.4108	1	20.96943	21.12187	0.07743	0.010777	0.16	0	0.010	0.098	0.060	#	#	#	#	#	#	0.380	13
2	34.7	34.70616	28.34207	6.357934	0.712418	-2.25188	0.5	35.84247	32.09227	0.075537	0.007891	0.3	0	0.006	0.091	0.046	#	#	#	#	#	#	0.304	3
3	33.4	33.45557	34.33926	-0.93926	0.7415	-2.28892	1	35.54724	34.33926	0.080683	0.016687	0.05	0	0.007	0.220	0.120	#	#	#	#	#	#	0.337	13
4	36.2	36.16096	29.87428	6.325723	0.812425	-1.96081	0.5	33.84469	31.85949	0.210545	0.008144	0.02	0	0.008	0.112	0.079	#	#	#	#	#	#	0.404	13
5	28.7	28.66165	26.59363	2.106372	0.827288	-2.15701	0.5	27.30924	26.95144	0.119276	0.001894	0.12	0	0.007	0.247	0.056	#	#	#	#	#	#	0.309	13
6	22.9	22.85388	20.57296	2.327042	0.819207	-2.04399	0.5	21.00549	20.78923	0.052651	0.008135	0.12	0	0.010	0.176	0.063	#	#	#	#	#	#	0.396	13

File is exported in `xlsx` format as:

```
GXGB.xlsx      (if spatial_weights=True,  alpha_wt_type= 'varying')
LW_GXGB.xlsx   (if spatial_weights=True,  alpha_wt_type= 'fixed', alpha_wt=1)
L_GXGB.xlsx    (if spatial_weights=False, alpha_wt_type= 'fixed', alpha_wt=1)
```

Step 7 Predict (unseen data)

```python
## Step 7 Predict (unseen data)
# Input data to predict
DataPredict = pd.read_csv('PredictData.csv')
CoordsPredict= pd.read_csv('PredCoords.csv')
# predict
Output_PredictGXGBoost= gx.predict_gxgb(DataPredict, CoordsPredict, Coords, Output_GXGB_LocalModel, alpha_wt = 0.5, alpha_wt_type = 'varying')
```

Run step 7 to use `geoxgboost` for prediction in unseen data. It needs a trained model that already exists. Have in mind to use local trained models with the same parameters as those specified in the function. For example, if prediction is used with `alpha_wt=0.5` and `alpha_wt_type='varying'`, the `Output_GXGB_LocalModel` should have been created with the same parameters in step 6.

Console output:

```
=============Predict Geographical-XGBoost ==========================
      Y_PRED
0   28.285767
1   24.500759
2   24.333361
3   22.468391
4   20.419663
5   20.256575
6   17.181181
7   15.742832
8   18.224682
Results have been saved in xlsx format at the specified path
======================================================================
```

```
Output_PredictGXGBoost = {DataFrame: (9, 1)...View as DataFrame
  T = {DataFrame: (1, 9)} [0, 1, 2, 3, 4, 5, 6, :...View as DataFrame
  Y_PRED = {Series: (9,)} (0, 28.2857666015625) ...View as Series
```

File is exported in `csv` format as `Predict_results.csv`