splot Documentation

Release 1.1.3

pysal developers

Mar 23, 2020

CONTENTS:

Release 1.1.3

Date Mar 23, 2020

splot provides *PySAL* users with a lightweight visualization interface to explore their data and quickly iterate through static and dynamic visualisations.

CHAPTER

ONE

INSTALLATION

1.1 Installing dependencies

splot is compatible with Python 3.6 and 3.7 and depends on GeoPandas 0.4.0 or later and matplotlib 2.2.2 or later. Please make sure that you are operating in a Python 3 environment.

splot also uses

- numpy
- seaborn
- mapclassify
- Ipywidgets

Depending on your spatial analysis workflow and the PySAL objects you would like to visualize, splot relies on:

PySAL >=2.0

or the installation of separate packages found in the PySAL stack:

- esda
- libpysal
- spreg
- giddy

1.2 Installing the newest release

There are two ways of accessing splot. First, splot is installed with the PySAL 2.0 metapackage through:

```
`$ pip install -U pysal`
```

or

```
`$ conda install -c conda-forge pysal`
```

Second, splot can be installed as a separate package. If you are using Anaconda, install splot via the conda utility:

```
`$ conda install -c conda-forge splot`
```

Otherwise, you can install splot from PyPI with pip:

```
`$ pip install splot`
```

1.3 Troubleshooting

Most common installation errors are due to splot's dependency on GeoPandas.

It often helps to first install GeoPandas separately from conda-forge with:

```
`$ conda install --channel conda-forge geopandas`
```

before installing splot (preferably also from conda, alternatively from pip).

For more information on troubleshooting the installation of GeoPandas with pip, see the GeoPandas docuemntation.

It is also possible to install splot with a later Python version (>3.7) through the separate installation of GeoPandas or through installation with conda-forge. (Note that splot is currently only tested for Python version 3.6 and 3.7)

1.4 Installing the development version

Potentially, you might want to use the newest features in the development version of splot on github - pysal/splot while have not been incorporated in the Pypi released version. You can achieve that by installing pysal/splot by running the following from a command shell:

```
pip install git+https://github.com/pysal/splot.git
```

You can also fork the pysal/splot repo and create a local clone of your fork. By making changes to your local clone and submitting a pull request to pysal/splot, you can contribute to the splot development.

CHAPTER

TWO

API REFERENCE

2.1 splot.giddy

Provides visualisations for the Geospatial Distribution Dynamics - *giddy* module. *giddy* provides a tool for space–time analytics that consider the role of space in the evolution of distributions over time.

2.1.1 Directional LISA analytics

dynamic_lisa_heatmap(rose[, p, ax])	Heatmap indicating significant transition of LISA values over time inbetween Moran Scatterplot quadrants
dynamic_lisa_rose(rose[, attribute, ax])	Plot dynamic LISA values in a rose diagram.
dynamic_lisa_vectors(rose[, ax, arrows])	Plot vectors of positional transition of LISA values in Moran scatterplot
dynamic_lisa_composite(rose, gdf[, p, figsize])	Composite visualisation for dynamic LISA values over two points in time.
dynamic_lisa_composite_explore(rose, gdf[, ...])	Interactive exploration of dynamic LISA values for different dates in a dataframe.

splot.giddy.dynamic_lisa_heatmap

splot.giddy.**dynamic_lisa_heatmap**(*rose*, *p=0.05*, *ax=None*, ***kwargs*)

Heatmap indicating significant transition of LISA values over time inbetween Moran Scatterplot quadrants

Parameters

rose [giddy.directional.Rose instance] A `Rose` object, which contains (among other attributes) LISA values at two points in time, and a method to perform inference on those.

p [float, optional] The p-value threshold for significance. Default =0.05

ax [Matplotlib Axes instance, optional] If given, the figure will be created inside this axis. Default =None.

****kwargs** [keyword arguments, optional] Keywords used for creating and designing the heatmap. These are passed on to *seaborn.heatmap()*. See *seaborn* documentation for valid keywords. Note: "Start time" refers to *y1* in *Y = np.array([y1, y2]).T* with *giddy.Rose(Y, w, k=5)*, "End time" referst to *y2*.

Returns

fig [Matplotlib Figure instance] Heatmap figure

ax [matplotlib Axes instance] Axes in which the figure is plotted

Examples

```
>>> import geopandas as gpd
>>> import pandas as pd
>>> from libpysal.weights.contiguity import Queen
>>> from libpysal import examples
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from giddy.directional import Rose
>>> from splot.giddy import dynamic_lisa_heatmap
```

get csv and shp files

```
>>> shp_link = examples.get_path('us48.shp')
>>> df = gpd.read_file(shp_link)
>>> income_table = pd.read_csv(examples.get_path("usjoin.csv"))
```

calculate relative values

```
>>> for year in range(1969, 2010):
...     income_table[str(year) + '_rel'] = (
...         income_table[str(year)] / income_table[str(year)].mean())
```

merge to one gdf

```
>>> gdf = df.merge(income_table,left_on='STATE_NAME',right_on='Name')
```

retrieve spatial weights and data for two points in time

```
>>> w = Queen.from_dataframe(gdf)
>>> w.transform = 'r'
>>> y1 = gdf['1969_rel'].values
>>> y2 = gdf['2000_rel'].values
```

calculate rose Object

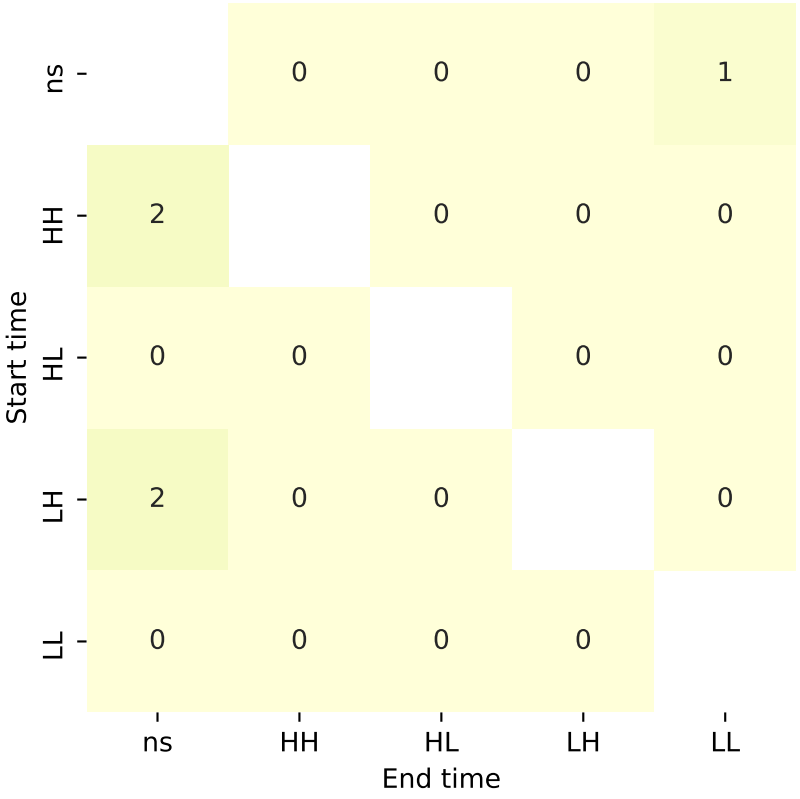
```
>>> Y = np.array([y1, y2]).T
>>> rose = Rose(Y, w, k=5)
```

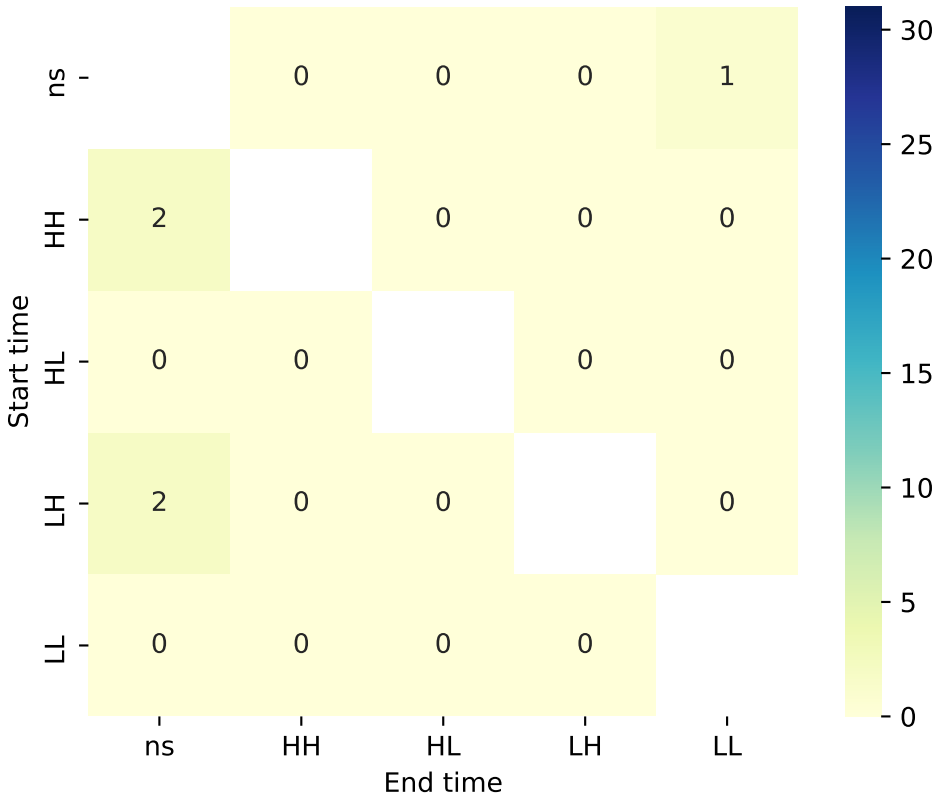
plot

```
>>> dynamic_lisa_heatmap(rose)
>>> plt.show()
```

customize plot

```
>>> dynamic_lisa_heatmap(rose, cbar='GnBu')
>>> plt.show()
```


Start time
ns
HH
HL
LH
LL
0 0 0 1
2 0 0 0
0 0 0 0
2 0 0 0
0 0 0 0
ns
HH
HL
LH
LL
End time


Start time
ns
HH
HL
LH
LL
0 0 0 1
2 0 0 0
0 0 0 0
2 0 0 0
0 0 0 0
ns
HH
HL
LH
LL
End time
30
25
20
15
10
5
0

splot.giddy.dynamic_lisa_rose

splot.giddy.**dynamic_lisa_rose**(*rose*, *attribute=None*, *ax=None*, ***kwargs*)

Plot dynamic LISA values in a rose diagram.

Parameters

rose [giddy.directional.Rose instance] A `Rose` object, which contains (among other attributes) LISA values at two points in time, and a method to perform inference on those.

attribute [(n,) ndarray, optional] Points will be colored by chosen attribute values. Variable to specify colors of the colorbars. Default =None.

ax [Matplotlib Axes instance, optional] If given, the figure will be created inside this axis. Default =None. Note: This axis should have a polar projection.

****kwargs** [keyword arguments, optional] Keywords used for creating and designing the *matplotlib.pyplot.scatter()*. Note: 'c' and 'color' cannot be passed when attribute is not None.

Returns

fig [Matplotlib Figure instance] LISA rose plot figure

ax [matplotlib Axes instance] Axes in which the figure is plotted

Examples

```
>>> import geopandas as gpd
>>> import pandas as pd
>>> from libpysal.weights.contiguity import Queen
>>> from libpysal import examples
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from giddy.directional import Rose
>>> from splot.giddy import dynamic_lisa_rose
```

get csv and shp files

```
>>> shp_link = examples.get_path('us48.shp')
>>> df = gpd.read_file(shp_link)
>>> income_table = pd.read_csv(examples.get_path("usjoin.csv"))
```

calculate relative values

```
>>> for year in range(1969, 2010):
...     income_table[str(year) + '_rel'] = (
...         income_table[str(year)] / income_table[str(year)].mean())
```

merge to one gdf

```
>>> gdf = df.merge(income_table,left_on='STATE_NAME',right_on='Name')
```

retrieve spatial weights and data for two points in time

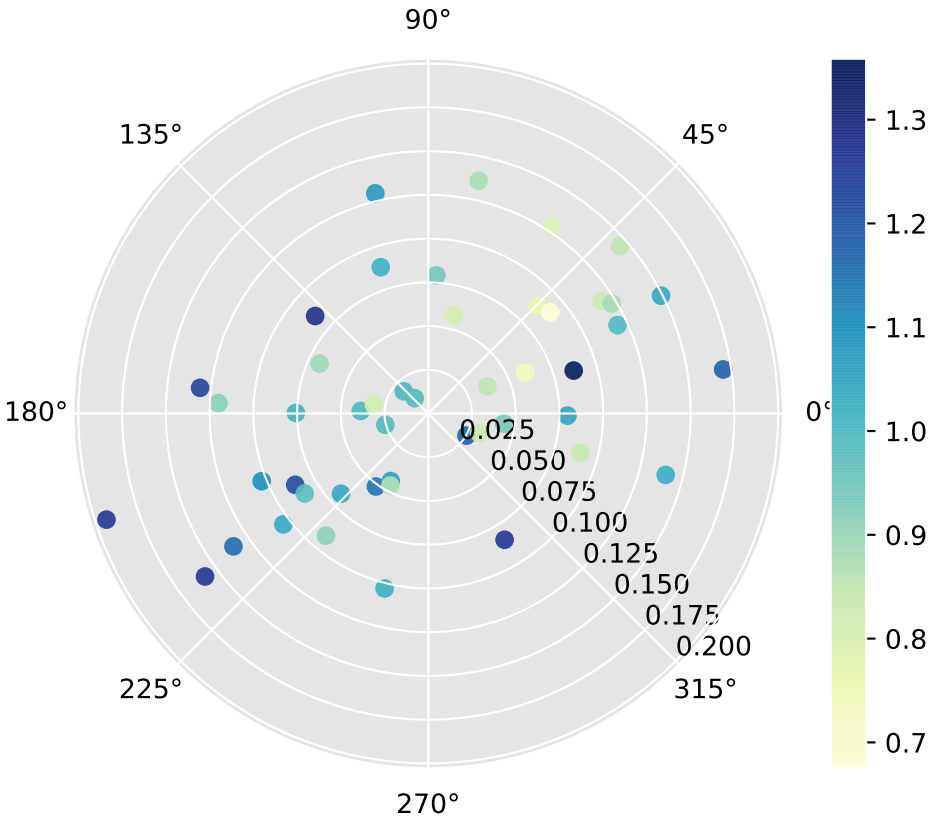
```
>>> w = Queen.from_dataframe(gdf)
>>> w.transform = 'r'
>>> y1 = gdf['1969_rel'].values
>>> y2 = gdf['2000_rel'].values
```

calculate rose Object

```
>>> Y = np.array([y1, y2]).T
>>> rose = Rose(Y, w, k=5)
```

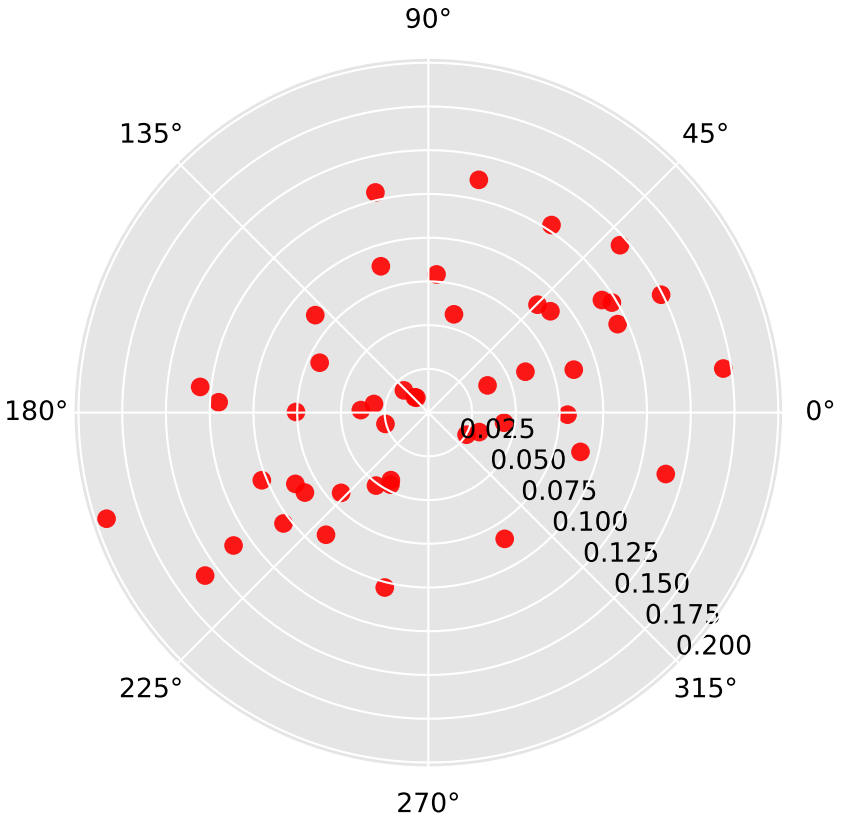
plot

```
>>> dynamic_lisa_rose(rose, attribute=y1)
>>> plt.show()
```



customize plot

```
>>> dynamic_lisa_rose(rose, c='r')
>>> plt.show()
```


90°
135°
45°
180°
0°
225°
315°
270°
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

splot.giddy.dynamic_lisa_vectors

splot.giddy.**dynamic_lisa_vectors**(*rose*, *ax=None*, *arrows=True*, ***kwargs*)

Plot vectors of positional transition of LISA values in Moran scatterplot

Parameters

rose [giddy.directional.Rose instance] A `Rose` object, which contains (among other attributes) LISA values at two points in time, and a method to perform inference on those.

ax [Matplotlib Axes instance, optional] If given, the figure will be created inside this axis. Default =None.

arrows [boolean, optional] If True show arrowheads of vectors. Default =True

****kwargs** [keyword arguments, optional] Keywords used for creating and designing the *matplotlib.pyplot.plot()* Note: 'c' and 'color' cannot be passed when attribute is not None.

Returns

fig [Matplotlib Figure instance] Figure of dynamic LISA vectors

ax [matplotlib Axes instance] Axes in which the figure is plotted

Examples

```
>>> import geopandas as gpd
>>> import pandas as pd
>>> from libpysal.weights.contiguity import Queen
>>> from libpysal import examples
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

```
>>> from giddy.directional import Rose
>>> from splot.giddy import dynamic_lisa_vectors
```

get csv and shp files

```
>>> shp_link = examples.get_path('us48.shp')
>>> df = gpd.read_file(shp_link)
>>> income_table = pd.read_csv(examples.get_path("usjoin.csv"))
```

calculate relative values

```
>>> for year in range(1969, 2010):
...     income_table[str(year) + '_rel'] = (
...         income_table[str(year)] / income_table[str(year)].mean())
```

merge to one gdf

```
>>> gdf = df.merge(income_table,left_on='STATE_NAME',right_on='Name')
```

retrieve spatial weights and data for two points in time

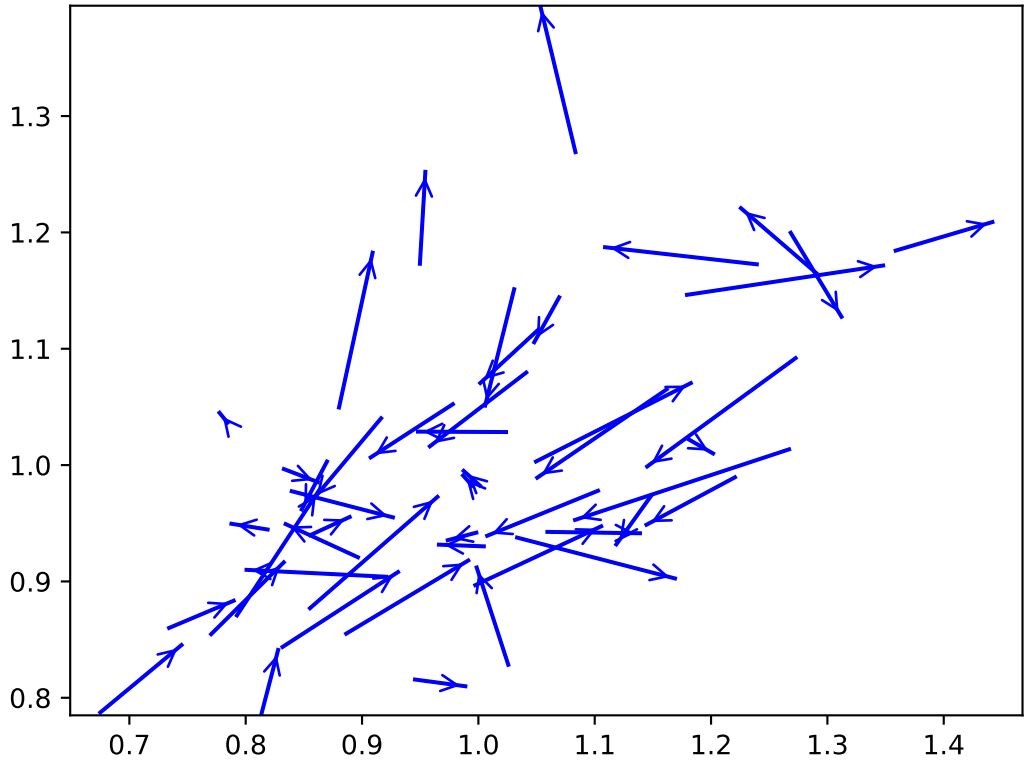
```
>>> w = Queen.from_dataframe(gdf)
>>> w.transform = 'r'
>>> y1 = gdf['1969_rel'].values
>>> y2 = gdf['2000_rel'].values
```

calculate rose Object

```
>>> Y = np.array([y1, y2]).T
>>> rose = Rose(Y, w, k=5)
```

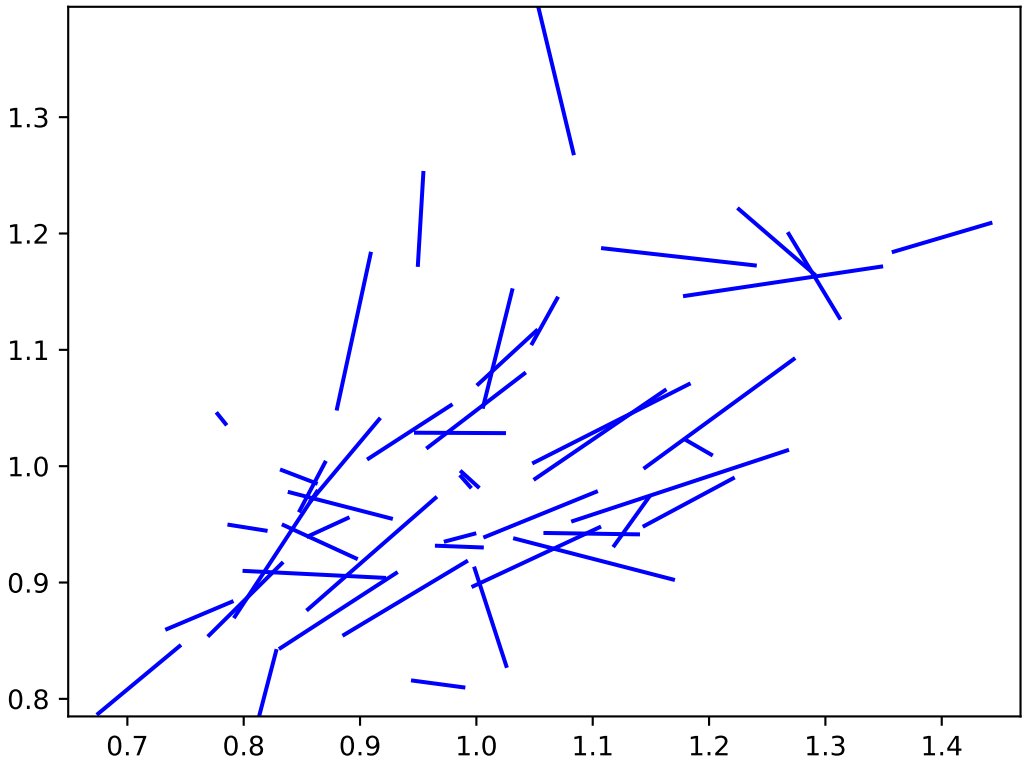
plot

```
>>> dynamic_lisa_vectors(rose)
>>> plt.show()
```



customize plot

```
>>> dynamic_lisa_vectors(rose, arrows=False, c='r')
>>> plt.show()
```


1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1.4

splot.giddy.dynamic_lisa_composite

splot.giddy.**dynamic_lisa_composite**(*rose*, *gdf*, *p=0.05*, *figsize=(13, 10)*)

Composite visualisation for dynamic LISA values over two points in time. Includes dynamic lisa heatmap, dynamic lisa rose plot, and LISA cluster plots for both, compared points in time.

Parameters

rose [giddy.directional.Rose instance] A `Rose` object, which contains (among other attributes) LISA values at two points in time, and a method to perform inference on those.

gdf [geopandas dataframe instance] The GeoDataFrame containing information and polygons to plot.

p [float, optional] The p-value threshold for significance. Default =0.05.

figsize: tuple, optional W, h of figure. Default =(13,10)

Returns

fig [Matplotlib Figure instance] Dynamic lisa composite figure.

axs [matplotlib Axes instance] Axes in which the figure is plotted.

Examples

```
>>> import geopandas as gpd
>>> import pandas as pd
>>> from libpysal.weights.contiguity import Queen
>>> from libpysal import examples
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from giddy.directional import Rose
>>> from splot.giddy import dynamic_lisa_composite
```

get csv and shp files

```
>>> shp_link = examples.get_path('us48.shp')
>>> df = gpd.read_file(shp_link)
>>> income_table = pd.read_csv(examples.get_path("usjoin.csv"))
```

calculate relative values

```
>>> for year in range(1969, 2010):
...     income_table[str(year) + '_rel'] = (
...         income_table[str(year)] / income_table[str(year)].mean())
```

merge to one gdf

```
>>> gdf = df.merge(income_table,left_on='STATE_NAME',right_on='Name')
```

retrieve spatial weights and data for two points in time

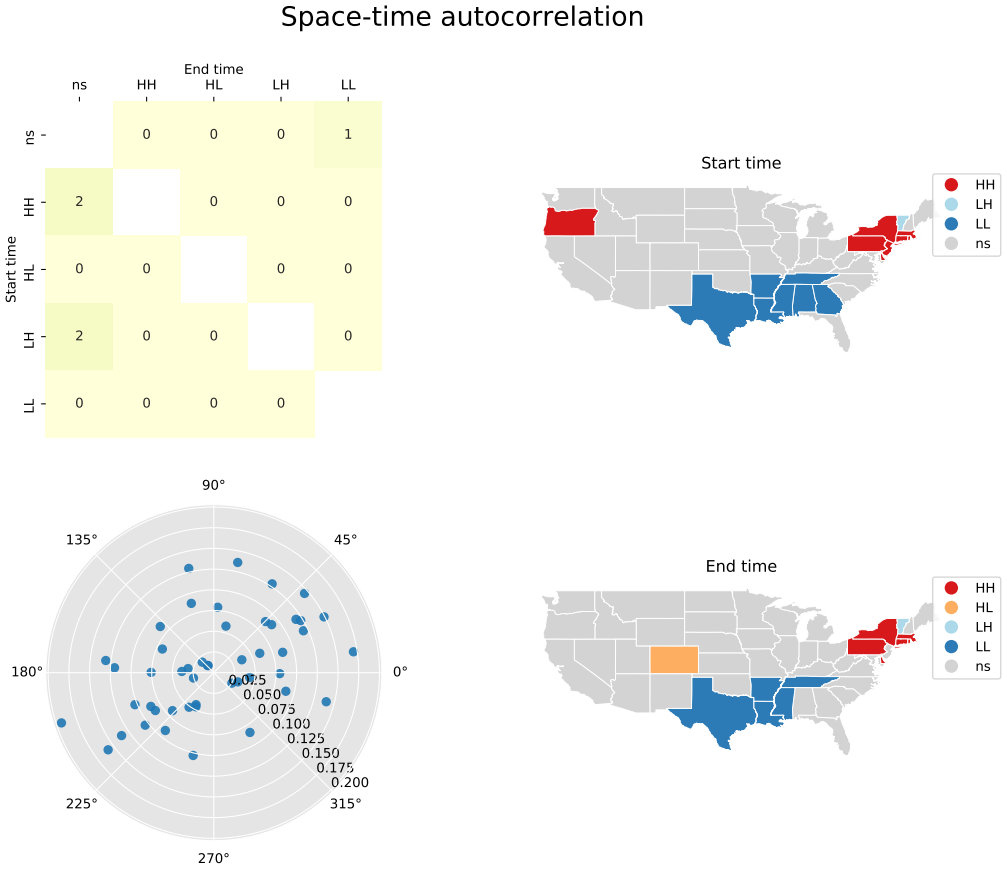
```
>>> w = Queen.from_dataframe(gdf)
>>> w.transform = 'r'
>>> y1 = gdf['1969_rel'].values
>>> y2 = gdf['2000_rel'].values
```

calculate rose Object

```
>>> Y = np.array([y1, y2]).T
>>> rose = Rose(Y, w, k=5)
```

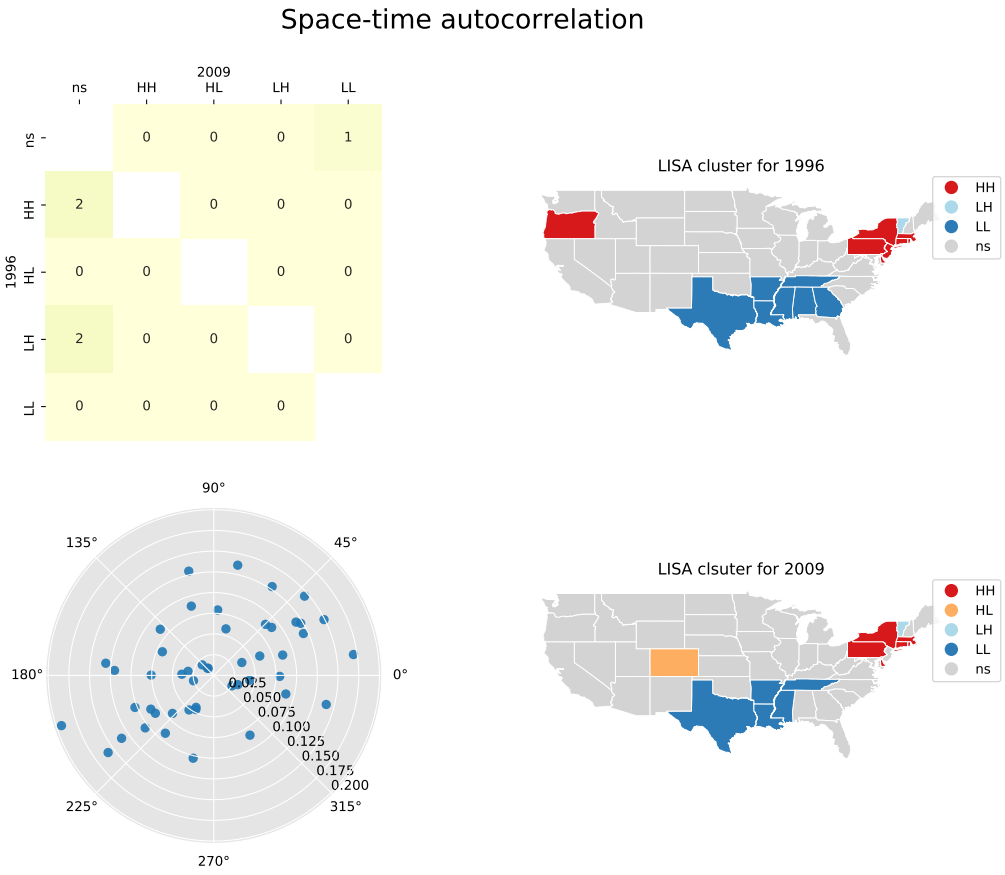
plot

```
>>> dynamic_lisa_composite(rose, gdf)
>>> plt.show()
```



customize plot

```
>>> fig, axs = dynamic_lisa_composite(rose, gdf)
>>> axs[0].set_ylabel('1996')
>>> axs[0].set_xlabel('2009')
>>> axs[1].set_title('LISA cluster for 1996')
>>> axs[3].set_title('LISA clsuter for 2009')
>>> plt.show()
```


Space-time autocorrelation
2009
ns
HH
HL
LH
LL
1996
ns
HH
HL
LH
LL
0 0 0 1
2 0 0 0
0 0 0 0
2 0 0 0
0 0 0 0
LISA cluster for 1996
HH
LH
LL
ns
90°
135°
45°
180°
0°
225°
315°
270°
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
LISA clsuter for 2009
HH
HL
LH
LL
ns

splot.giddy.dynamic_lisa_composite_explore

splot.giddy.**dynamic_lisa_composite_explore**(*rose*, *gdf*, *pattern=''*, *p=0.05*, *figsize=(13, 10)*)

Interactive exploration of dynamic LISA values for different dates in a dataframe. Note: only possible in jupyter notebooks

Parameters

rose [giddy.directional.Rose instance] A `Rose` object, which contains (among other attributes) weights to calculate *esda.moran.Moran_local* values

gdf [geopandas dataframe instance] The Dataframe containing information and polygons to plot.

pattern [str, optional] Option to extract all columns ending with a specific pattern. Only extracted columns will be used for comparison.

p [float, optional] The p-value threshold for significance. Default =0.05

figsize: tuple, optional W, h of figure. Default =(13,10)

Returns

None

Examples

Note: this function creates Jupyter notebook widgets, so is meant only to run in a notebook.

```
>>> import geopandas as gpd
>>> import pandas as pd
>>> from libpysal.weights.contiguity import Queen
>>> from libpysal import examples
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

If you want to see figures embedded inline in a Jupyter notebook, add a line `%matplotlib inline` at the top of your notebook.

```
>>> from giddy.directional import Rose
>>> from splot.giddy import dynamic_lisa_composite_explore
```

get csv and shp files

```
>>> shp_link = examples.get_path('us48.shp')
>>> df = gpd.read_file(shp_link)
>>> income_table = pd.read_csv(examples.get_path("usjoin.csv"))
```

calculate relative values

```
>>> for year in range(1969, 2010):
...     income_table[str(year) + '_rel'] = (
...         income_table[str(year)] / income_table[str(year)].mean())
```

merge to one gdf

```
>>> gdf = df.merge(income_table,left_on='STATE_NAME',right_on='Name')
```

retrieve spatial weights and data for two points in time

```
>>> w = Queen.from_dataframe(gdf)
>>> w.transform = 'r'
>>> y1 = gdf['1969_rel'].values
>>> y2 = gdf['2000_rel'].values
```

calculate rose Object

```
>>> Y = np.array([y1, y2]).T
>>> rose = Rose(Y, w, k=5)
```

plot

```
>>> fig = dynamic_lisa_composite_explore(rose, gdf, pattern='rel')
>>> # plt.show()
```

2.2 splot.esda

Provides visualisations for the *esda* subpackage. *esda* provides tools for exploratory spatial data analysis that consider the role of space in a distribution of attribute values.

2.2.1 Moran analytics

moran_scatterplot(moran[, zstandard, p, ...])	Moran Scatterplot
plot_moran_simulation(moran[, aspect_equal, ...])	Global Moran's I simulated reference distribution.
plot_moran(moran[, zstandard, aspect_equal, ...])	Global Moran's I simulated reference distribution and scatterplot.
plot_moran_bv_simulation(moran_bv[, ax, ...])	Bivariate Moran's I simulated reference distribution.
plot_moran_bv(moran_bv[, aspect_equal, ...])	Bivariate Moran's I simulated reference distribution and scatterplot.
lisa_cluster(moran_loc, gdf[, p, ax, ...])	Create a LISA Cluster map
plot_local_autocorrelation(moran_loc, gdf, ...)	Produce three-plot visualisation of Moran Scatteprlot, LISA cluster and Choropleth maps, with Local Moran region and quadrant masking
moran_facet(moran_matrix[, figsize, ...])	Moran Facet visualization.

splot.esda.moran_scatterplot

splot.esda.**moran_scatterplot**(*moran*, *zstandard=True*, *p=None*, *aspect_equal=True*, *ax=None*, *scatter_kwds=None*, *fitline_kwds=None*)

Moran Scatterplot

Parameters

moran [esda.moran instance] Values of Moran's I Global, Bivariate and Local Autocorrelation Statistics

zstandard [bool, optional] If True, Moran Scatterplot will show z-standardized attribute and spatial lag values. Default =True.

p [float, optional] If given, the p-value threshold for significance for Local Autocorrelation analysis. Points will be colored by significance. By default it will not be colored. Default =None.

aspect_equal [bool, optional] If True, Axes will show the same aspect or visual proportions for Moran Scatterplot.

ax [Matplotlib Axes instance, optional] If given, the Moran plot will be created inside this axis. Default =None.

scatter_kwds [keyword arguments, optional] Keywords used for creating and designing the scatter points. Default =None.

fitline_kwds [keyword arguments, optional] Keywords used for creating and designing the moran fitline. Default =None.

Returns

fig [Matplotlib Figure instance] Moran scatterplot figure

ax [matplotlib Axes instance] Axes in which the figure is plotted

Examples

Imports

```
>>> import matplotlib.pyplot as plt
>>> from libpysal.weights.contiguity import Queen
>>> from libpysal import examples
>>> import geopandas as gpd
>>> from esda.moran import (Moran, Moran_BV,
...                         Moran_Local, Moran_Local_BV)
>>> from splot.esda import moran_scatterplot
```

Load data and calculate weights

```
>>> link_to_data = examples.get_path('Guerry.shp')
>>> gdf = gpd.read_file(link_to_data)
>>> x = gdf['Suicids'].values
>>> y = gdf['Donatns'].values
>>> w = Queen.from_dataframe(gdf)
>>> w.transform = 'r'
```

Calculate esda.moran Objects

```
>>> moran = Moran(y, w)
>>> moran_bv = Moran_BV(y, x, w)
>>> moran_loc = Moran_Local(y, w)
>>> moran_loc_bv = Moran_Local_BV(y, x, w)
```

Plot

```
>>> fig, axs = plt.subplots(2, 2, figsize=(10,10),
...                         subplot_kw={'aspect': 'equal'})
>>> moran_scatterplot(moran, p=0.05, ax=axs[0,0])
>>> moran_scatterplot(moran_loc, p=0.05, ax=axs[1,0])
>>> moran_scatterplot(moran_bv, p=0.05, ax=axs[0,1])
>>> moran_scatterplot(moran_loc_bv, p=0.05, ax=axs[1,1])
>>> plt.show()
```

splot.esda.plot_moran_simulation

splot.esda.**plot_moran_simulation**(*moran*, *aspect_equal=True*, *ax=None*, *fitline_kwds=None*, ***kwargs*)

Global Moran's I simulated reference distribution.

Parameters

moran [esda.moran.Moran instance] Values of Moran's I Global Autocorrelation Statistics

aspect_equal [bool, optional] If True, Axes of Moran Scatterplot will show the same aspect or visual proportions.

ax [Matplotlib Axes instance, optional] If given, the Moran plot will be created inside this axis. Default =None.

fitline_kwds [keyword arguments, optional] Keywords used for creating and designing the vertical moran fitline. Default =None.

****kwargs** [keyword arguments, optional] Keywords used for creating and designing the figure, passed to seaborn.kdeplot.

Returns

fig [Matplotlib Figure instance] Simulated reference distribution figure

ax [matplotlib Axes instance] Axes in which the figure is plotted

Examples

Imports

```
>>> import matplotlib.pyplot as plt
>>> from libpysal.weights.contiguity import Queen
>>> from libpysal import examples
>>> import geopandas as gpd
>>> from esda.moran import Moran
>>> from splot.esda import plot_moran_simulation
```

Load data and calculate weights

```
>>> link_to_data = examples.get_path('Guerry.shp')
>>> gdf = gpd.read_file(link_to_data)
>>> y = gdf['Donatns'].values
>>> w = Queen.from_dataframe(gdf)
>>> w.transform = 'r'
```

Calculate Global Moran

```
>>> moran = Moran(y, w)
```

plot

```
>>> plot_moran_simulation(moran)
>>> plt.show()
```

customize plot

```
>>> plot_moran_simulation(moran, fitline_kwds=dict(color='#4393c3'))
>>> plt.show()
```

splot.esda.plot_moran

splot.esda.**plot_moran**(*moran*, *zstandard=True*, *aspect_equal=True*, *scatter_kwds=None*, *fitline_kwds=None*, ***kwargs*)

Global Moran's I simulated reference distribution and scatterplot.

Parameters

moran [esda.moran.Moran instance] Values of Moran's I Global Autocorrelation Statistics

zstandard [bool, optional] If True, Moran Scatterplot will show z-standardized attribute and spatial lag values. Default =True.

aspect_equal [bool, optional] If True, Axes of Moran Scatterplot will show the same aspect or visual proportions.

scatter_kwds [keyword arguments, optional] Keywords used for creating and designing the scatter points. Default =None.

fitline_kwds [keyword arguments, optional] Keywords used for creating and designing the moran fitline and vertical fitline. Default =None.

****kwargs** [keyword arguments, optional] Keywords used for creating and designing the figure, passed to seaborne.kdeplot.

Returns

fig [Matplotlib Figure instance] Moran scatterplot and reference distribution figure

ax [matplotlib Axes instance] Axes in which the figure is plotted

Examples

Imports

```
>>> import matplotlib.pyplot as plt
>>> from libpysal.weights.contiguity import Queen
>>> from libpysal import examples
>>> import geopandas as gpd
>>> from esda.moran import Moran
>>> from splot.esda import plot_moran
```

Load data and calculate weights

```
>>> link_to_data = examples.get_path('Guerry.shp')
>>> gdf = gpd.read_file(link_to_data)
>>> y = gdf['Donatns'].values
>>> w = Queen.from_dataframe(gdf)
>>> w.transform = 'r'
```

Calculate Global Moran

```
>>> moran = Moran(y, w)
```

plot

```
>>> plot_moran(moran)
>>> plt.show()
```

customize plot

```
>>> plot_moran(moran, zstandard=False,
...            fitline_kwds=dict(color='#4393c3'))
>>> plt.show()
```

splot.esda.plot_moran_bv_simulation

splot.esda.**plot_moran_bv_simulation**(*moran_bv*, *ax=None*, *aspect_equal=True*, *fitline_kwds=None*, ***kwargs*)

Bivariate Moran's I simulated reference distribution.

Parameters

moran_bv [esda.moran.Moran_BV instance] Values of Bivariate Moran's I Autocorrelation Statistics

ax [Matplotlib Axes instance, optional] If given, the Moran plot will be created inside this axis. Default =None.

aspect_equal [bool, optional] If True, Axes of Moran Scatterplot will show the same aspect or visual proportions.

fitline_kwds [keyword arguments, optional] Keywords used for creating and designing the vertical moran fitline. Default =None.

****kwargs** [keyword arguments, optional] Keywords used for creating and designing the figure, passed to seaborne.kdeplot.

Returns

fig [Matplotlib Figure instance] Bivariate moran reference distribution figure

ax [matplotlib Axes instance] Axes in which the figure is plotted

Examples

Imports

```
>>> import matplotlib.pyplot as plt
>>> from libpysal.weights.contiguity import Queen
>>> from libpysal import examples
>>> import geopandas as gpd
>>> from esda.moran import Moran_BV
>>> from splot.esda import plot_moran_bv_simulation
```

Load data and calculate weights

```
>>> link_to_data = examples.get_path('Guerry.shp')
>>> gdf = gpd.read_file(link_to_data)
>>> x = gdf['Suicids'].values
>>> y = gdf['Donatns'].values
>>> w = Queen.from_dataframe(gdf)
>>> w.transform = 'r'
```

Calculate Bivariate Moran

```
>>> moran_bv = Moran_BV(x, y, w)
```

plot

```
>>> plot_moran_bv_simulation(moran_bv)
>>> plt.show()
```

customize plot

```
>>> plot_moran_bv_simulation(moran_bv,
...                          fitline_kwds=dict(color='#4393c3'))
>>> plt.show()
```

splot.esda.plot_moran_bv

splot.esda.**plot_moran_bv**(*moran_bv*, *aspect_equal=True*, *scatter_kwds=None*, *fitline_kwds=None*, ***kwargs*)

Bivariate Moran's I simulated reference distribution and scatterplot.

Parameters

moran_bv [esda.moran.Moran_BV instance] Values of Bivariate Moran's I Autocorrelation Statistics

aspect_equal [bool, optional] If True, Axes of Moran Scatterplot will show the same aspect or visual proportions.

scatter_kwds [keyword arguments, optional] Keywords used for creating and designing the scatter points. Default =None.

fitline_kwds [keyword arguments, optional] Keywords used for creating and designing the moran fitline and vertical fitline. Default =None.

****kwargs** [keyword arguments, optional] Keywords used for creating and designing the figure, passed to seaborne.kdeplot.

Returns

fig [Matplotlib Figure instance] Bivariate moran scatterplot and reference distribution figure

ax [matplotlib Axes instance] Axes in which the figure is plotted

Examples

Imports

```
>>> import matplotlib.pyplot as plt
>>> from libpysal.weights.contiguity import Queen
>>> from libpysal import examples
>>> import geopandas as gpd
>>> from esda.moran import Moran_BV
>>> from splot.esda import plot_moran_bv
```

Load data and calculate weights

```
>>> link_to_data = examples.get_path('Guerry.shp')
>>> gdf = gpd.read_file(link_to_data)
>>> x = gdf['Suicids'].values
>>> y = gdf['Donatns'].values
>>> w = Queen.from_dataframe(gdf)
>>> w.transform = 'r'
```

Calculate Bivariate Moran

```
>>> moran_bv = Moran_BV(x, y, w)
```

plot

```
>>> plot_moran_bv(moran_bv)
>>> plt.show()
```

customize plot

```
>>> plot_moran_bv(moran_bv, fitline_kwds=dict(color='#4393c3'))
>>> plt.show()
```

splot.esda.lisa_cluster

splot.esda.**lisa_cluster**(*moran_loc*, *gdf*, *p=0.05*, *ax=None*, *legend=True*, *legend_kwds=None*, ***kwargs*)

Create a LISA Cluster map

Parameters

moran_loc [esda.moran.Moran_Local or Moran_Local_BV instance] Values of Moran's Local Autocorrelation Statistic

gdf [geopandas dataframe instance] The Dataframe containing information to plot. Note that *gdf* will be modified, so calling functions should use a copy of the user provided *gdf*. (either using gdf.assign() or gdf.copy())

p [float, optional] The p-value threshold for significance. Points will be colored by significance.

ax [matplotlib Axes instance, optional] Axes in which to plot the figure in multiple Axes layout. Default = None

legend [boolean, optional] If True, legend for maps will be depicted. Default = True

legend_kwds [dict, optional] Dictionary to control legend formatting options. Example: `legend_kwds={'loc': 'upper left', 'bbox_to_anchor': (0.92, 1.05)}` Default = None

****kwargs** [keyword arguments, optional] Keywords designing and passed to geopandas.GeoDataFrame.plot().

Returns

fig [matplotlip Figure instance] Figure of LISA cluster map

ax [matplotlib Axes instance] Axes in which the figure is plotted

Examples

Imports

```
>>> import matplotlib.pyplot as plt
>>> from libpysal.weights.contiguity import Queen
>>> from libpysal import examples
>>> import geopandas as gpd
>>> from esda.moran import Moran_Local
>>> from splot.esda import lisa_cluster
```

Data preparation and statistical analysis

```
>>> link = examples.get_path('Guerry.shp')
>>> gdf = gpd.read_file(link)
>>> y = gdf['Donatns'].values
>>> w = Queen.from_dataframe(gdf)
>>> w.transform = 'r'
>>> moran_loc = Moran_Local(y, w)
```

Plotting

```
>>> fig = lisa_cluster(moran_loc, gdf)
>>> plt.show()
```

splot.esda.plot_local_autocorrelation

splot.esda.**plot_local_autocorrelation**(*moran_loc*, *gdf*, *attribute*, *p=0.05*, *region_column=None*, *mask=None*, *mask_color='#636363'*, *quadrant=None*, *aspect_equal=True*, *legend=True*, *scheme='Quantiles'*, *cmap='YlGnBu'*, *figsize=(15, 4)*, *scatter_kwds=None*, *fitline_kwds=None*)

Produce three-plot visualisation of Moran Scatteprlot, LISA cluster and Choropleth maps, with Local Moran region and quadrant masking

Parameters

moran_loc [esda.moran.Moran_Local or Moran_Local_BV instance] Values of Moran's Local Autocorrelation Statistic

gdf [geopandas dataframe] The Dataframe containing information to plot the two maps.

attribute [str] Column name of attribute which should be depicted in Choropleth map.

p [float, optional] The p-value threshold for significance. Points and polygons will be colored by significance. Default = 0.05.

region_column: string, optional Column name containing mask region of interest. Default = None

mask: str, optional Identifier or name of the region to highlight. Default = None

mask_color: str, optional Color of mask. Default = '#636363'

quadrant [int, optional] Quadrant 1-4 in scatterplot masking values in LISA cluster and Choropleth maps. Default = None

aspect_equal [bool, optional] If True, Axes of Moran Scatterplot will show the same aspect or visual proportions.

figsize: tuple, optional W, h of figure. Default = (15,4)

legend: boolean, optional If True, legend for maps will be depicted. Default = True

scheme: str, optional Name of PySAL classifier to be used. Default = 'Quantiles'

cmap: str, optional Name of matplotlib colormap used for plotting the Choropleth. Default = 'YlGnBu'

scatter_kwds [keyword arguments, optional] Keywords used for creating and designing the scatter points. Default =None.

fitline_kwds [keyword arguments, optional] Keywords used for creating and designing the moran fitline in the scatterplot. Default =None.

Returns

fig [Matplotlib figure instance] Moran Scatterplot, LISA cluster map and Choropleth.

axs [list of Matplotlib axes] Lisat of Matplotlib axes plotted.

Examples

Imports

```
>>> import matplotlib.pyplot as plt
>>> from libpysal.weights.contiguity import Queen
>>> from libpysal import examples
>>> import geopandas as gpd
>>> from esda.moran import Moran_Local
>>> from splot.esda import plot_local_autocorrelation
```

Data preparation and analysis

```
>>> link = examples.get_path('Guerry.shp')
>>> gdf = gpd.read_file(link)
>>> y = gdf['Donatns'].values
>>> w = Queen.from_dataframe(gdf)
>>> w.transform = 'r'
>>> moran_loc = Moran_Local(y, w)
```

Plotting with quadrant mask and region mask

```
>>> fig = plot_local_autocorrelation(moran_loc, gdf, 'Donatns', p=0.05,
...                                  region_column='Dprtmnt',
...                                  mask=['Ain'], quadrant=1)
>>> plt.show()
```

splot.esda.moran_facet

splot.esda.**moran_facet**(*moran_matrix*, *figsize=(16, 12)*, *scatter_bv_kwds=None*, *fitline_bv_kwds=None*, *scatter_glob_kwds={'color': '#737373'}*, *fitline_glob_kwds=None*)

Moran Facet visualization. Includes BV Morans and Global Morans on the diagonal.

Parameters

moran_matrix [esda.moran.Moran_BV_matrix instance] Dictionary of Moran_BV objects

figsize [tuple, optional] W, h of figure. Default =(16,12)

scatter_bv_kwds [keyword arguments, optional] Keywords used for creating and designing the scatter points of off-diagonal Moran_BV plots. Default =None.

fitline_bv_kwds [keyword arguments, optional] Keywords used for creating and designing the moran fitline of off-diagonal Moran_BV plots. Default =None.

scatter_glob_kwds [keyword arguments, optional] Keywords used for creating and designing the scatter points of diagonal Moran plots. Default =None.

fitline_glob_kwds [keyword arguments, optional] Keywords used for creating and designing the moran fitline of diagonal Moran plots. Default =None.

Returns

fig [Matplotlib Figure instance] Bivariate Moran Local scatterplot figure

axarr [matplotlib Axes instance] Axes in which the figure is plotted

Examples

Imports

```
>>> import matplotlib.pyplot as plt
>>> import libpysal as lp
>>> import numpy as np
>>> import geopandas as gpd
>>> from esda.moran import Moran_BV_matrix
>>> from splot.esda import moran_facet
```

Load data and calculate Moran Local statistics

```
>>> f = gpd.read_file(lp.examples.get_path("sids2.dbf"))
>>> varnames = ['SIDR74',  'SIDR79',  'NWR74',  'NWR79']
>>> vars = [np.array(f[var]) for var in varnames]
>>> w = lp.io.open(lp.examples.get_path("sids2.gal")).read()
>>> moran_matrix = Moran_BV_matrix(vars,  w,  varnames = varnames)
```

Plot

```
>>> fig, axarr = moran_facet(moran_matrix)
>>> plt.show()
```

Customize plot

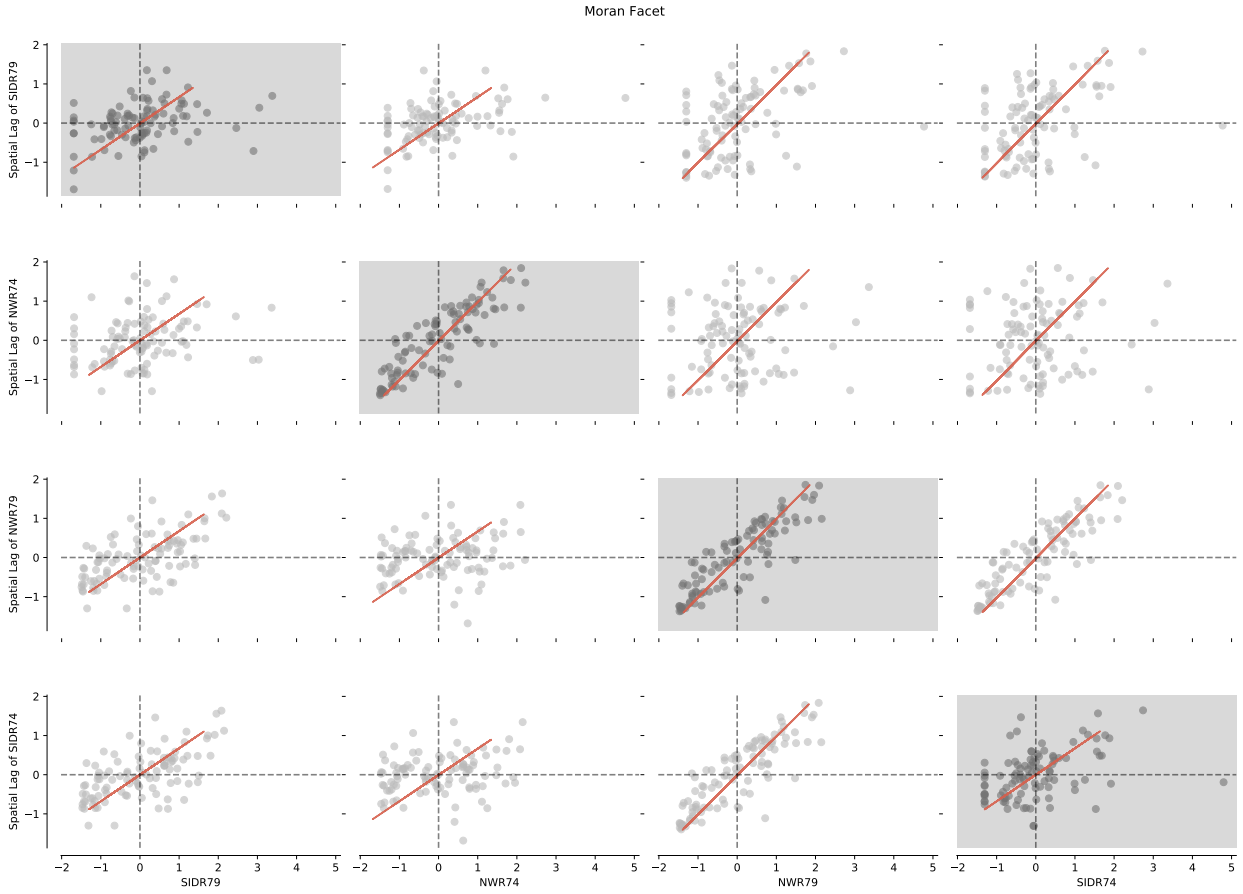
```
>>> fig, axarr = moran_facet(moran_matrix,
...                          fitline_bv_kwds=dict(color='#4393c3'))
>>> plt.show()
```

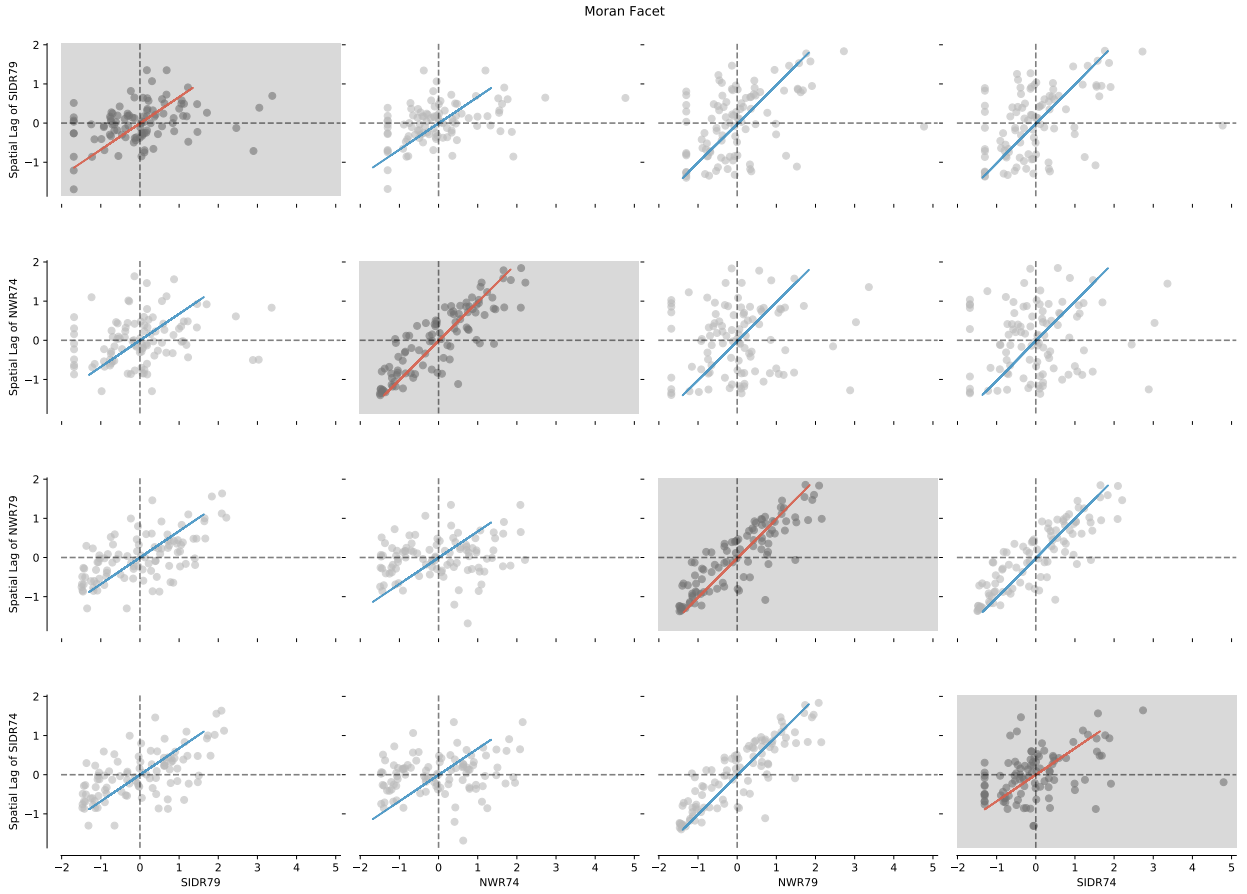
2.3 splot.libpysal

Provides visualisations for all core components of Python Spatial Analysis Library in *libpysal*.

2.3.1 libpysal weights

plot_spatial_weights(w, gdf[, indexed_on, ...])	Plot spatial weights network.


Moran Facet
Spatial Lag of SIDR79
Spatial Lag of NWR74
Spatial Lag of NWR79
Spatial Lag of SIDR74
SIDR79
NWR74
NWR79
SIDR74


Moran Facet
Spatial Lag of SIDR79
Spatial Lag of NWR74
Spatial Lag of NWR79
Spatial Lag of SIDR74
SIDR79
NWR74
NWR79
SIDR74

splot.libpysal.plot_spatial_weights

splot.libpysal.**plot_spatial_weights**(*w*, *gdf*, *indexed_on=None*, *ax=None*, *figsize=(10, 10)*, *node_kws=None*, *edge_kws=None*, *nonplanar_edge_kws=None*)

Plot spatial weights network. NOTE: Additionally plots *w.non_planar_joins* if *libpysal.weights.util.nonplanar_neighbors()* was applied.

Parameters

w [libpysal.W object] Values of libpysal weights object.

gdf [geopandas dataframe] The original shapes whose topological relations are modelled in W.

indexed_on [str, optional] Column of gdf which the weights object uses as an index. Default =None, so the geodataframe's index is used.

ax [matplotlib axis, optional] Axis on which to plot the weights. Default =None, so plots on the current figure.

figsize [tuple, optional] W, h of figure. Default =(10,10)

node_kws [keyword argument dictionary, optional] Dictionary of keyword arguments to send to pyplot.scatter, which provide fine-grained control over the aesthetics of the nodes in the plot. Default =None.

edge_kws [keyword argument dictionary, optional] Dictionary of keyword arguments to send to pyplot.plot, which provide fine-grained control over the aesthetics of the edges in the plot. Default =None.

nonplanar_edge_kws [keyword argument dictionary, optional] Dictionary of keyword arguments to send to pyplot.plot, which provide fine-grained control over the aesthetics of the edges from *weights.non_planar_joins* in the plot. Default =None.

Returns

fig [matplotlip Figure instance] Figure of spatial weight network.

ax [matplotlib Axes instance] Axes in which the figure is plotted.

Examples

Imports

```
>>> from libpysal.weights.contiguity import Queen
>>> import geopandas as gpd
>>> import libpysal
>>> from libpysal import examples
>>> import matplotlib.pyplot as plt
>>> from splot.libpysal import plot_spatial_weights
```

Data preparation and statistical analysis

```
>>> gdf = gpd.read_file(examples.get_path('map_RS_BR.shp'))
>>> weights = Queen.from_dataframe(gdf)
>>> wnp = libpysal.weights.util.nonplanar_neighbors(weights, gdf)
```

Plot weights

```
>>> plot_spatial_weights(weights, gdf)
>>> plt.show()
```

Plot corrected weights

```
>>> plot_spatial_weights(wnp, gdf)
>>> plt.show()
```

2.4 splot.mapping

Provides Choropleth visualizations and mapping utilities.

2.4.1 Value-by-Alpha maps

value_by_alpha_cmap(x, y[, cmap, ...])	Calculates Value by Alpha rgba values
vba_choropleth(x_var, y_var, gdf[, cmap, ...])	Value by Alpha Choropleth
vba_legend(rgb_bins, alpha_bins, cmap[, ax])	Creates Value by Alpha heatmap used as choropleth legend.
mapclassify_bin(y, classifier[, k, pct, ...])	Classify your data with *pysal.mapclassify* Note: Input parameters are dependent on classifier used.

splot.mapping.value_by_alpha_cmap

splot.mapping.**value_by_alpha_cmap**(*x*, *y*, *cmap='GnBu'*, *revert_alpha=False*, *divergent=False*)

Calculates Value by Alpha rgba values

Parameters

x [array] Variable determined by color

y [array] Variable determining alpha value

cmap [str or list of str] Matplotlib Colormap or list of colors used to create vba_layer

revert_alpha [bool, optional] If True, high y values will have a low alpha and low values will be transparent. Default =False.

divergent [bool, optional] Creates a divergent alpha array with high values at the extremes and low, transparent values in the middle of the input values.

Returns

rgba [ndarray (n,4)] RGBA colormap, where the alpha channel represents one attribute (x) and the rgb color the other attribute (y)

cmap [str or list of str] Original Matplotlib Colormap or list of colors used to create vba_layer

Examples

Imports

```
>>> from libpysal import examples
>>> import geopandas as gpd
>>> import matplotlib.pyplot as plt
>>> import matplotlib
>>> import numpy as np
>>> from splot.mapping import value_by_alpha_cmap
```

Load Example Data

```
>>> link_to_data = examples.get_path('columbus.shp')
>>> gdf = gpd.read_file(link_to_data)
>>> x = gdf['HOVAL'].values
>>> y = gdf['CRIME'].values
```

Create rgba values

```
>>> rgba, _ = value_by_alpha_cmap(x, y)
```

Create divergent rgba and change Colormap

```
>>> div_rgba, _ = value_by_alpha_cmap(x, y, cmap='seismic', divergent=True)
```

Create rgba values with reverted alpha values

```
>>> rev_rgba, _  = value_by_alpha_cmap(x, y, cmap='RdBu', revert_alpha=True)
```

splot.mapping.vba_choropleth

splot.mapping.**vba_choropleth**(*x_var*, *y_var*, *gdf*, *cmap='GnBu'*, *divergent=False*, *revert_alpha=False*, *alpha_mapclassify=None*, *rgb_mapclassify=None*, *ax=None*, *legend=False*)

Value by Alpha Choropleth

Parameters

x_var [string or array] The name of variable in gdf determined by color or an array of values determined by color.

y_var [string or array] The name of variable in gdf determining alpha value or an array of values determined by color.

gdf [geopandas dataframe instance] The Dataframe containing information to plot.

cmap [str or list of str] Matplotlib Colormap or list of colors used to create vba_layer

divergent [bool, optional] Creates a divergent alpha array with high values at the extremes and low, transparent values in the middle of the input values.

revert_alpha [bool, optional] If True, high y values will have a low alpha and low values will be transparent. Default = False.

alpha_mapclassify [dict] Keywords used for binning input values and classifying alpha values with *mapclassify*. Note: valid keywords are eg. dict(classifier='quantiles', k=5, hinge=1.5). For other options check *splot.mapping.mapclassify_bin*.

rgb_mapclassify [dict] Keywords used for binning input values and classifying rgb values with *mapclassify*. Note: valid keywords are eg.g dict(classifier='quantiles', k=5, hinge=1.5).For other options check *splot.mapping.mapclassify_bin*.

ax [matplotlib Axes instance, optional] Axes in which to plot the figure in multiple Axes layout. Default = None

legend [bool, optional] Adds a legend. Note: currently only available if data is classified, hence if *alpha_mapclassify* and *rgb_mapclassify* are used.

Returns

fig [matplotlip Figure instance] Figure of Value by Alpha choropleth

ax [matplotlib Axes instance] Axes in which the figure is plotted

Examples

Imports

```
>>> from libpysal import examples
>>> import geopandas as gpd
>>> import matplotlib.pyplot as plt
>>> import matplotlib
>>> import numpy as np
>>> from splot.mapping import vba_choropleth
```

Load Example Data

```
>>> link_to_data = examples.get_path('columbus.shp')
>>> gdf = gpd.read_file(link_to_data)
```

Plot a Value-by-Alpha map

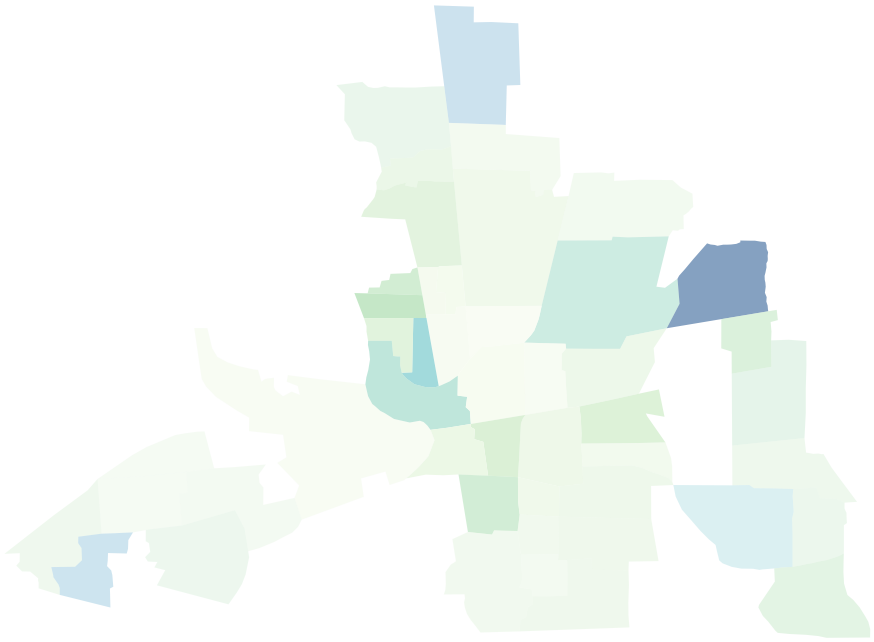
```
>>> fig, _ = vba_choropleth('HOVAL', 'CRIME', gdf)
>>> plt.show()
```

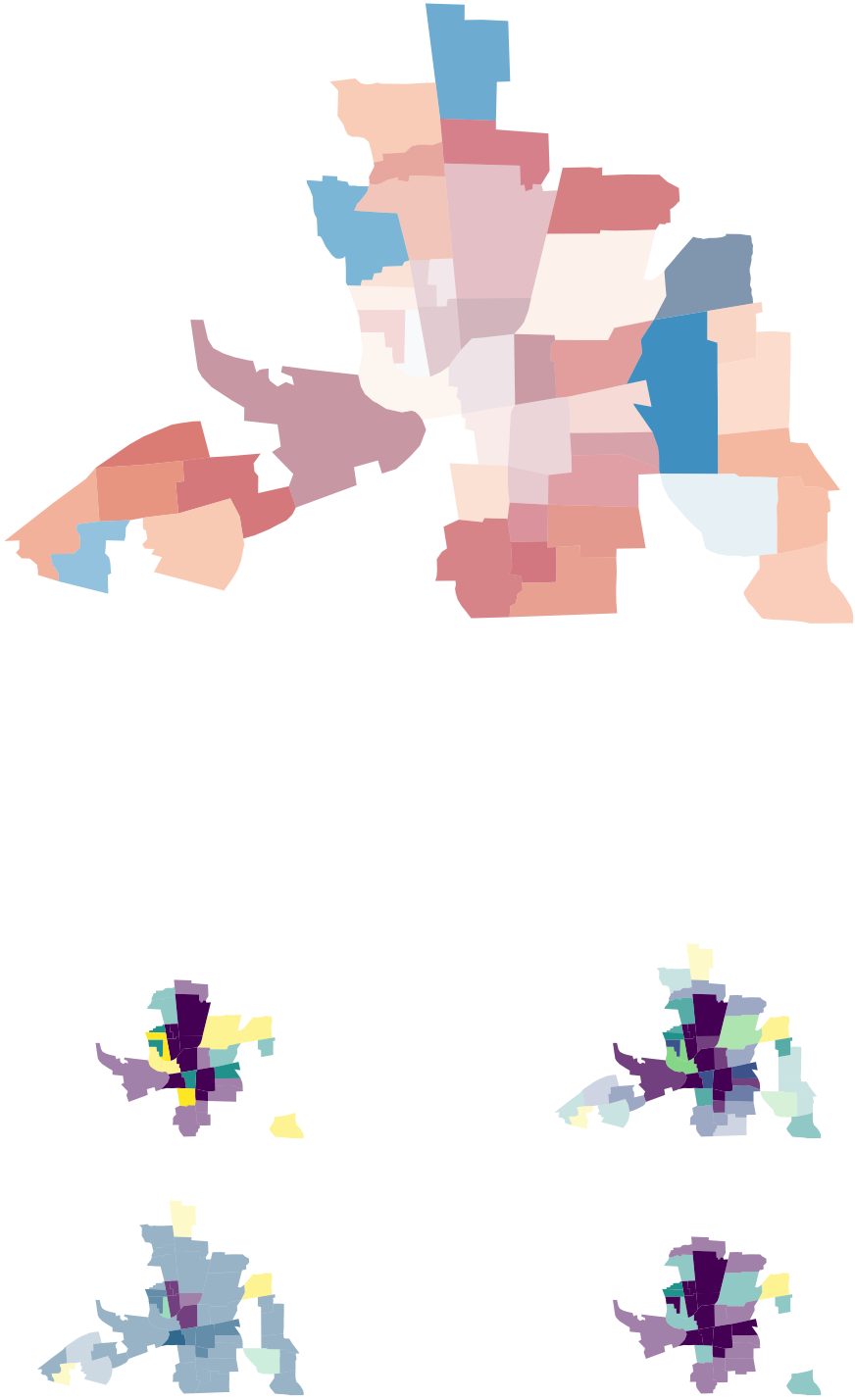
Plot a Value-by-Alpha map with reverted alpha values

```
>>> fig, _ = vba_choropleth('HOVAL', 'CRIME', gdf, cmap='RdBu',
...                         revert_alpha=True)
>>> plt.show()
```

Plot a Value-by-Alpha map with classified alpha and rgb values

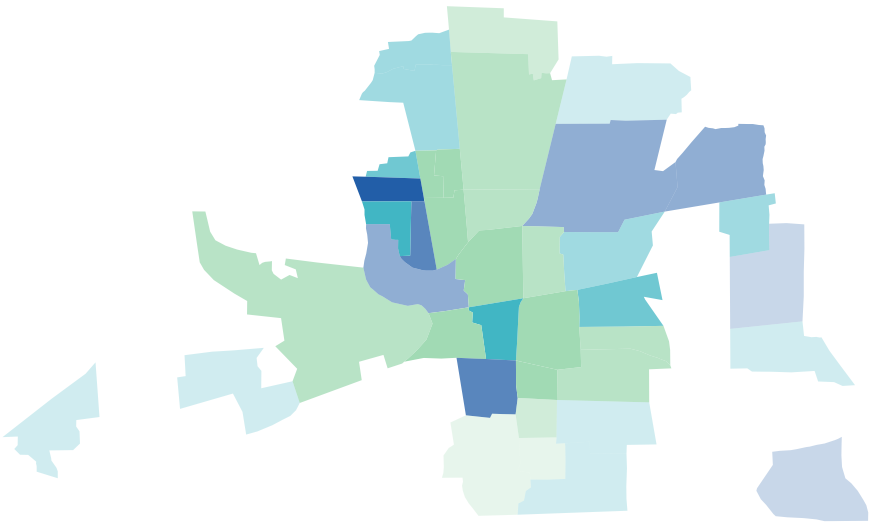
```
>>> fig, axs = plt.subplots(2,2, figsize=(20,10))
>>> vba_choropleth('HOVAL', 'CRIME', gdf, cmap='viridis', ax = axs[0,0],
...                rgb_mapclassify=dict(classifier='quantiles', k=3),
...                alpha_mapclassify=dict(classifier='quantiles', k=3))
>>> vba_choropleth('HOVAL', 'CRIME', gdf, cmap='viridis', ax = axs[0,1],
...                rgb_mapclassify=dict(classifier='natural_breaks'),
...                alpha_mapclassify=dict(classifier='natural_breaks'))
>>> vba_choropleth('HOVAL', 'CRIME', gdf, cmap='viridis', ax = axs[1,0],
...                rgb_mapclassify=dict(classifier='std_mean'),
...                alpha_mapclassify=dict(classifier='std_mean'))
>>> vba_choropleth('HOVAL', 'CRIME', gdf, cmap='viridis', ax = axs[1,1],
...                rgb_mapclassify=dict(classifier='fisher_jenks', k=3),
...                alpha_mapclassify=dict(classifier='fisher_jenks', k=3))
>>> plt.show()
```





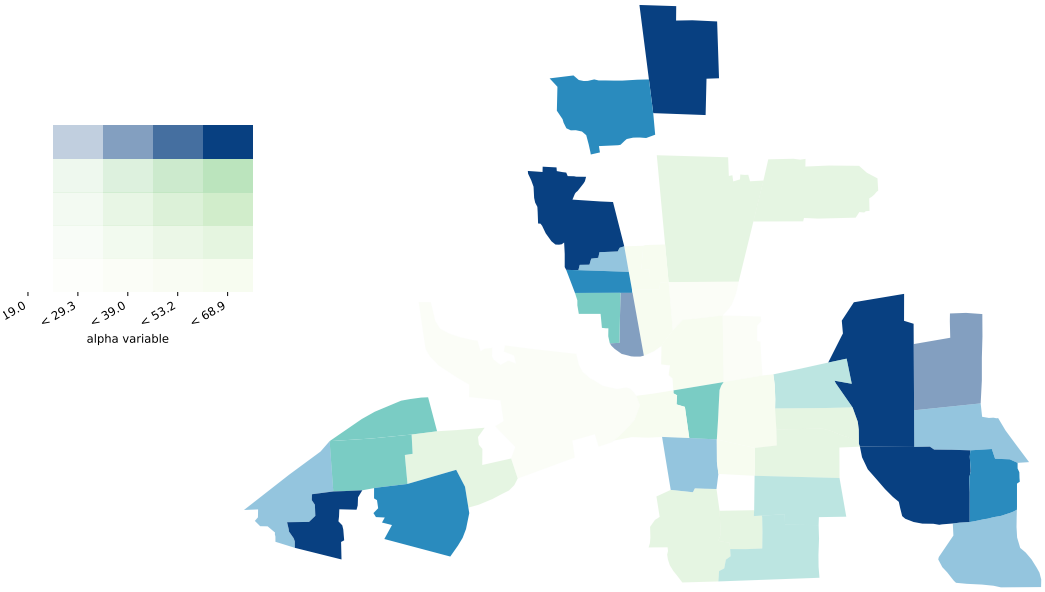
Pass in a list of colors instead of a cmap

```
>>> color_list = ['#a1dab4','#41b6c4','#225ea8']
>>> vba_choropleth('HOVAL', 'CRIME', gdf, cmap=color_list,
...                rgb_mapclassify=dict(classifier='quantiles', k=3),
...                alpha_mapclassify=dict(classifier='quantiles'))
>>> plt.show()
```



Add a legend and use divergent alpha values

```
>>> fig = plt.figure(figsize=(15,10))
>>> ax = fig.add_subplot(111)
>>> vba_choropleth('HOVAL', 'CRIME', gdf, divergent=True,
...                alpha_mapclassify=dict(classifier='quantiles', k=5),
...                rgb_mapclassify=dict(classifier='quantiles', k=5),
...                legend=True, ax=ax)
>>> plt.show()
```



splot.mapping.vba_legend

`splot.mapping.`**`vba_legend`**(*rgb_bins*, *alpha_bins*, *cmap*, *ax=None*)

Creates Value by Alpha heatmap used as choropleth legend.

Parameters

rgb_bins [pysal.mapclassify instance] Object of classified values used for rgb. Can be created with *mapclassify_bin()* or *pysal.mapclassify*.

alpha_bins [pysal.mapclassify instance] Object of classified values used for alpha. Can be created with *mapclassify_bin()* or *pysal.mapclassify*.

ax [matplotlib Axes instance, optional] Axes in which to plot the figure in multiple Axes layout. Default = None

Returns

fig [matplotlip Figure instance] Figure of Value by Alpha heatmap

ax [matplotlib Axes instance] Axes in which the figure is plotted

Examples

Imports

```
>>> from libpysal import examples
>>> import geopandas as gpd
>>> import matplotlib.pyplot as plt
>>> import matplotlib
>>> import numpy as np
>>> from splot.mapping import vba_legend, mapclassify_bin
```

Load Example Data

```
>>> link_to_data = examples.get_path('columbus.shp')
>>> gdf = gpd.read_file(link_to_data)
>>> x = gdf['HOVAL'].values
>>> y = gdf['CRIME'].values
```

Classify your data

```
>>> rgb_bins = mapclassify_bin(x, 'quantiles')
>>> alpha_bins = mapclassify_bin(y, 'quantiles')
```

Plot your legend

```
>>> fig, _ = vba_legend(rgb_bins, alpha_bins, cmap='RdBu')
>>> plt.show()
```

splot.mapping.mapclassify_bin

splot.mapping.**mapclassify_bin**(*y, classifier, k=5, pct=[1, 10, 50, 90, 99, 100], hinge=1.5, multiples=[-2, -1, 1, 2], mindiff=0, initial=100, bins=None*)

Classify your data with *pysal.mapclassify* Note: Input parameters are dependent on classifier used.

Parameters

y [array] (n,1), values to classify

classifier [str] pysal.mapclassify classification scheme

k [int, optional] The number of classes. Default=5.

pct [array, optional] Percentiles used for classification with *percentiles*. Default=[1,10,50,90,99,100]

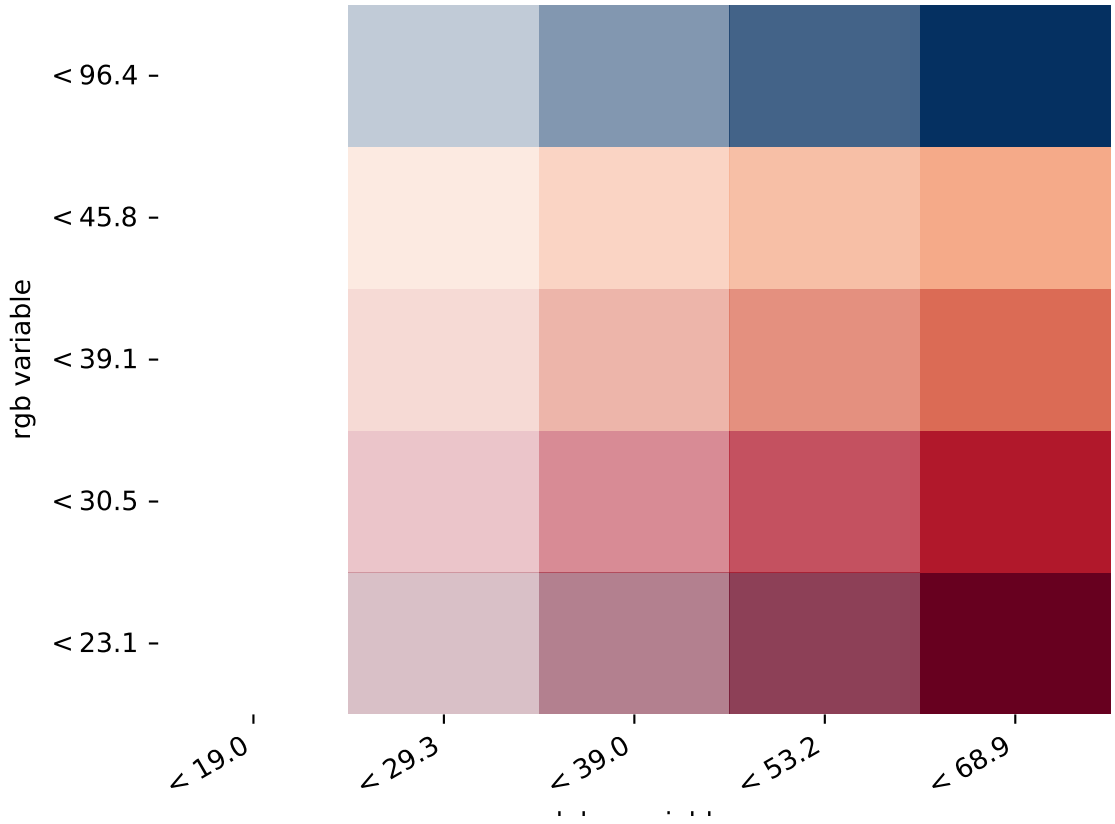
hinge [float, optional] Multiplier for IQR when *BoxPlot* classifier used. Default=1.5.

multiples [array, optional] The multiples of the standard deviation to add/subtract from the sample mean to define the bins using *std_mean*. Default=[-2,-1,1,2].

mindiff [float, optional] The minimum difference between class breaks if using *maximum_breaks* classifier. Deafult =0.

initial [int] Number of initial solutions to generate or number of runs when using *natural_breaks* or *max_p_classifier*. Default =100. Note: setting initial to 0 will result in the quickest calculation of bins.

bins [array, optional] (k,1), upper bounds of classes (have to be monotically increasing) if using *user_defined* classifier. Default =None, Example =[20, max(y)].


rgb variable
< 96.4
< 45.8
< 39.1
< 30.5
< 23.1
< 19.0
< 29.3
< 39.0
< 53.2
< 68.9

Returns

bins [pysal.mapclassify instance] Object containing bin ids for each observation (.yb), upper bounds of each class (.bins), number of classes (.k) and number of onservations falling in each class (.counts)

Note: Supported classifiers include: quantiles, box_plot, euqal_interval, fisher_jenks, headtail_breaks, jenks_caspall, jenks_caspall_forced, max_p_classifier, maximum_breaks, natural_breaks, percentiles, std_mean, user_defined

Examples

Imports

```
>>> from libpysal import examples
>>> import geopandas as gpd
>>> from splot.mapping import mapclassify_bin
```

Load Example Data

```
>>> link_to_data = examples.get_path('columbus.shp')
>>> gdf = gpd.read_file(link_to_data)
>>> x = gdf['HOVAL'].values
```

Classify values by quantiles

```
>>> quantiles = mapclassify_bin(x, 'quantiles')
```

Classify values by box_plot and set hinge to 2

```
>>> box_plot = mapclassify_bin(x, 'box_plot', hinge=2)
```

2.4.2 Colormap utilities

shift_colormap(cmap[, start, midpoint, ...])	Function to offset the "center" of a colormap.
truncate_colormap(cmap[, minval, maxval, n])	Function to truncate a colormap by selecting a subset of the original colormap's values

splot.mapping.shift_colormap

splot.mapping.**shift_colormap**(*cmap*, *start=0*, *midpoint=0.5*, *stop=1.0*, *name='shiftedcmap'*)

Function to offset the "center" of a colormap. Useful for data with a negative min and positive max and you want the middle of the colormap's dynamic range to be at zero

Parameters

cmap [str or matplotlib.cm instance] colormap to be altered

start [float, optional] Offset from lowest point in the colormap's range. Should be between 0.0 and *midpoint*. Default =0.0 (no lower ofset).

midpoint [float, optional] The new center of the colormap.Should be between 0.0 and 1.0. In general, this should be 1 - vmax/(vmax + abs(vmin)). For example if your data range from -15.0 to +5.0 and you want the center of the colormap at 0.0, *midpoint* should be set to 1 -

5/(5 + 15)) or 0.75. Default =0.5 (no shift).

stop [float, optional] Offset from highets point in the colormap's range. Should be between *midpoint* and 1.0. Default =1.0 (no upper ofset).

name [str, optional] Name of the new colormap.

Returns

new_cmap [A new colormap that has been shifted.]

splot.mapping.truncate_colormap

splot.mapping.**truncate_colormap**(*cmap*, *minval=0.0*, *maxval=1.0*, *n=100*)

Function to truncate a colormap by selecting a subset of the original colormap's values

Parameters

cmap [str or matplotlib.cm instance] Colormap to be altered

minval [float, optional] Minimum value of the original colormap to include in the truncated colormap. Default =0.0.

maxval [Maximum value of the original colormap to] include in the truncated colormap. Default =1.0.

n [int, optional] Number of intervals between the min and max values for the gradient of the truncated colormap. Default =100.

Returns

new_cmap [A new colormap that has been shifted.]

CHAPTER

THREE

REFERENCES

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX