GeoHexViz

Release 1.0.0

Tony Abou Zeidan

Oct 21, 2021

CONTENTS:

CHAPTER

ONE

GEOHEXVIZ - PLOT BUILDER MODULE

class geohexviz.builder.**PlotBuilder**(*hexbin_layer: Optional[Dict[str, Any]] = None*, *regions: Optional[Dict[str, Dict[str, Any]]] = None*, *grids: Optional[Dict[str, Dict[str, Any]]] = None*, *outlines: Optional[Dict[str, Dict[str, Any]]] = None*, *points: Optional[Dict[str, Dict[str, Any]]] = None*, *use_templates: bool = True*)

This class contains a Builder implementation for visualizing Plotly Hex data.

add_grid(*name: str*, *data: Union[str, pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]*, *hex_resolution: Optional[int] = None*, *latitude_field: Optional[str] = None*, *longitude_field: Optional[str] = None*, *convex_simplify: bool = False*)

Adds a grid-type layer to the builder.

Grid-type layers should consist of Polygon-like or Point-like geometries.

Parameters

- **name** (*str*) – The name this layer is to be stored with
- **data** (*Union[str, DataFrame, GeoDataFrame]*) – The location of the data for this layer
- **hex_resolution** (*int*) – The hexagonal resolution to use for this layer (None->builder default)
- **latitude_field** (*str*) – The latitude column within the data
- **longitude_field** (*str*) – The longitude column within the data
- **convex_simplify** (*bool*) – Determines if the area the grid is to be placed over should be simplified or not

add_outline(*name: str*, *data: Union[str, pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]*, *latitude_field: Optional[str] = None*, *longitude_field: Optional[str] = None*, *as_boundary: bool = False*, *manager: Optional[Dict[str, Any]] = None*)

Adds a outline-type layer to the builder.

Parameters

- **name** (*str*) – The name this layer is to be stored with
- **data** (*Union[str, DataFrame, GeoDataFrame]*) – The location of the data for this layer
- **latitude_field** (*str*) – The latitude column of the data
- **longitude_field** (*str*) – The longitude column of the data
- **as_boundary** (*bool*) – Changes the data into one big boundary if true

- **manager** (*StrDict*) – Plotly properties for this layer

add_point(*name: str*, *data: Union[str, pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]*, *latitude_field: Optional[str] = None*, *longitude_field: Optional[str] = None*, *text_field: Optional[str] = None*, *manager: Optional[Dict[str, Any]] = None*)

Adds a point-type layer to the builder.

Ideally the layer's 'data' member should contain lat/long columns or point like geometry column. If the geometry column is present and contains no point like geometry, the geometry will be converted into a bunch of points.

Parameters

- **name** (*str*) – The name this layer is to be stored with
- **data** (*Union[str, DataFrame, GeoDataFrame]*) – The location of the data for this layer
- **latitude_field** (*str*) – The latitude column of the data
- **longitude_field** (*str*) – The longitude column of the data
- **text_field** (*str*) – The column containing text for data entries
- **manager** (*StrDict*) – Plotly properties for this layer

add_region(*name: str*, *data: Union[str, pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]*, *manager: Optional[Dict[str, Any]] = None*)

Adds a region-type layer to the builder.

Region-type layers should consist of Polygon-like geometries. Best results are read from a GeoDataFrame, or DataFrame.

Parameters

- **name** (*str*) – The name this layer is to be stored with
- **data** (*Union[str, DataFrame, GeoDataFrame]*) – The location of the data for this layer
- **manager** (*StrDict*) – The plotly properties for this layer.

adjust_colorbar_size(*width=700*, *height=450*, *t=20*, *b=20*)

Adjusts the color scale position of the color bar to match the plot area size.

Does not work.

adjust_focus(*on: str = 'hexbin'*, *center_on: bool = False*, *rotation_on: bool = True*, *ranges_on: bool = True*, *rot_buffer_lat: float = 0*, *rot_buffer_lon: float = 0*, *buffer_lat: tuple = (0, 0)*, *buffer_lon: tuple = (0, 0)*, *validate: bool = False*)

Focuses on layer(s) within the plot.

Collects the geometries of the queried layers in order to obtain a boundary to focus on.

In the future using a GeoSeries may be looked into for cleaner code.

Parameters

- **on** (*str*) – The query for the layer(s) to be focused on
- **center_on** (*bool*) – Whether or not to add a center component to the focus
- **rotation_on** (*bool*) – Whether or not to add a projection rotation to the focus
- **ranges_on** (*bool*) – Whether or not to add a lat axis, lon axis ranges to the focus

- **rot_buffer_lat** (*float*) – A number to add or subtract from the automatically calculated latitude (rotation)
- **rot_buffer_lon** (*float*) – A number to add or subtract from the automatically calculated longitude (rotation)
- **buffer_lat** (*Tuple[float, float]*) – A low and high bound to add and subtract from the lataxis range
- **buffer_lon** (*Tuple[float, float]*) – A low and high bound to add and subtract from the lonaxis range
- **validate** (*bool*) – Whether or not to validate the ranges

adjust_opacity(*alpha: Optional[float] = None*)
Conforms the opacity of the color bar of the hexbin layer to an alpha value.

The alpha value can be passed in as a parameter, otherwise it is taken from the marker.opacity property within the layer's manager.

Parameters **alpha** (*float*) – The alpha value to conform the color scale to

apply_to_query(*name: str*, *fn*, **args*, *allow_empty: bool = True*, ***kwargs*)
Applies a function to the layers within a query.

For advanced users and not to be used carelessly. The functions first argument must be the layer.

Parameters

- **name** (*str*) – The query of the layers to apply the function to
- **fn** (*Callable*) – The function to apply
- **allow_empty** (*bool*) – Whether to allow query arguments that retrieved empty results or not

auto_grid(*on: str = 'hexbin'*, *by_bounds: bool = False*, *hex_resolution: Optional[int] = None*)
Makes a grid over the queried layers.

Parameters

- **on** (*str*) – The query for the layers to have a grid generated over them
- **by_bounds** (*bool*) – Whether or not to treat the geometries as a single boundary
- **hex_resolution** (*int*) – The hexagonal resolution to use for the auto grid

static builder_from_dict(*builder_dict: Optional[Dict[str, Any]] = None*, ***kwargs*)
Makes a builder from a dictionary.

Parameters

- **builder_dict** (*StrDict*) – The dictionary to build from
- **kwargs** (**kwargs) – Keyword arguments for the builder

clear_figure()
Clears the figure of its current data.

clear_grid_manager()
Clears the manager of a region layer.

If the given name is none, clears all of the region managers.

clear_hexbin_manager()
Clears the manager of the hexbin layer.

clear_outline_manager(*name: Optional[str] = None*)
Clears the manager of a outline layer.

If the given name is none, clears all of the outline managers.

clear_point_manager(*name: Optional[str] = None*)
Clears the manager of a point layer.

If the given name is none, clears all of the point managers.

clear_region_manager(*name: Optional[str] = None*)
Clears the manager of a region layer.

If the given name is none, clears all of the region managers.

clip_layers(*clip: str*, *to: str*, *method: str = 'sjoin'*, *reduce_first: bool = True*, *operation: str = 'intersects'*)
Clips a query of layers to another layer. this function is experimental and may not always work as intended

There are two methods for this clipping: 1) sjoin -> Uses GeoPandas spatial join in order to clip geometries

that (intersect, are within, contain, etc.) the geometries acting as the clip.

2) **gpd -> Uses GeoPandas clip function in order to clip geometries** to the boundary of the geometries acting as the clip.

Parameters

- **clip** (*GeoDataFrame*) – The query for the layers that are to be clipped to another
- **to** (*GeoDataFrame*) – The query for the layers that are to be used as the boundary
- **method** (*str*) – The method to use when clipping, one of 'sjoin', 'gpd'
- **reduce_first** (*bool*) – Determines whether the geometries acting as the clip should be reduced first or not
- **operation** (*str*) – The operation to apply when using sjoin (spatial join operation)

discretize_scale(*scale_type: str = 'sequential'*, ***kwargs*)
Converts the color scale of the layer(s) to a discrete scale.

Parameters

- **scale_type** (*str*) – One of 'sequential', 'discrete' for the type of color scale being used
- **kwargs** (**kwargs) – Keyword arguments to be passed into the discretize functions

display(*clear_figure: bool = False*, ***kwargs*)
Displays the figure.

The figure is displayed via Plotly's show() function. Extensions may be needed.

Parameters

- **clear_figure** (*bool*) – Whether or not to clear the figure after this operation
- **kwargs** (**kwargs) – Keyword arguments for the show function

finalize(*plot_regions: bool = True*, *plot_grids: bool = True*, *plot_hexbin: bool = True*, *plot_outlines: bool = True*, *plot_points: bool = True*, *raise_errors: bool = False*)
Builds the final plot by adding traces in order.

Invokes the functions in the following order: 1) plot regions 2) plot grids 3) plot layer 4) plot outlines 5) plot points

In the future we should alter these functions to allow trace order implementation.

Parameters

- **plot_regions** (*bool*) – Whether or not to plot region layers
- **plot_grids** (*bool*) – Whether or not to plot grid layers
- **plot_hexbin** (*bool*) – Whether or not to plot the hexbin layer
- **plot_outlines** (*bool*) – Whether or not to plot outline layers
- **plot_points** (*bool*) – Whether or not to plot point layers
- **raise_errors** (*bool*) – Whether or not to raise errors related to empty layer collections

get_grid(*name: str*) → Dict[str, Any]
Retrieves a grid layer from the builder.

External version, returns a deepcopy.

Parameters **name** (*str*) – The name of the layer

Returns The retrieved layer

Return type StrDict

get_grids() → Dict[str, Dict[str, Any]]
Retrieves the grid layers from the builder.

External version, returns a deepcopy.

Returns The retrieved layers

Return type Dict[str, StrDict]

get_hexbin()
Retrieves the main layer.

External version, returns a deepcopy.

Returns The main layer

Return type StrDict

get_outline(*name: str*) → Dict[str, Any]
Retrieves a outline layer from the builder.

External version, returns a deepcopy.

Parameters **name** (*str*) – The name of the layer

Returns The retrieved layer

Return type StrDict

get_outlines() → Dict[str, Dict[str, Any]]
Retrieves the outline layers from the builder.

External version, returns a deepcopy.

Returns The retrieved layers

Return type Dict[str, StrDict]

get_plot_output_service() → str
Retrieves the current plot output service for the builder.

Returns The current plot output service

Return type str

get_plot_status() → *geohexviz.builder.PlotStatus*
Retrieves the status of the internal plot.

Returns The status of the plot

Return type *PlotStatus*

get_point(*name: str*) → Dict[str, Any]
Retrieves a point layer from the builder.

External version, returns a deepcopy.

Parameters **name** (*str*) – The name of the layer

Returns The retrieved layer

Return type StrDict

get_points() → Dict[str, Dict[str, Any]]
Retrieves the collection of point layers in the builder.

External version, returns a deepcopy.

Returns The point layers within the builder

Return type Dict[str, StrDict]

get_query_data(*name*)
PLOTTING FUNCTIONS

get_region(*name: str*) → Dict[str, Any]
Retrieves a region layer from the builder.

External version, returns a deepcopy.

Parameters **name** (*str*) – The name of the layer

Returns The retrieved layer

Return type StrDict

get_regions() → Dict[str, Dict[str, Any]]
Retrieves the region layers from the builder.

External version, returns a deepcopy.

Returns The retrieved layers

Return type Dict[str, StrDict]

logify_scale(***kwargs*)
Makes the scale of the hexbin layers logarithmic.

This function changes the tick values and tick text of the scale. The numerical values on the scale are the exponent of the tick text, i.e the text of 1 on the scale actually represents the value of zero, and the text of 1000 on the scale actually represents the value of 3.

Parameters **kwargs** (**kwargs) – Keyword arguments to be passed into logify functions

output(*filepath: Optional[str] = None*, *clear_figure: bool = False*, *crop_output: bool = False*, *percent_retain=None*, *keep_original: bool = False*, ***kwargs*)
Outputs the figure to a filepath.

The figure is output via Plotly's write_image() function. Plotly's Kaleido is required for this feature.

Parameters

- **filepath** (*str*) – The filepath to output the figure at (including filename and extension)

- **clear_figure** (*bool*) – Whether or not to clear the figure after this operation
- **crop_output** (*bool*) – Whether or not to crop the output figure (requires that extension be pdf, PdfCropMargins must be installed)
- **percent_retain** (*float, str, list*) – Percentage of margins to retain from crop (requires that extension be pdf, PdfCropMargins must be installed)
- **keep_original** (*bool*) – Whether or not to keep the original figure (requires that extension be pdf, PdfCropMargins must be installed)
- **kwargs** (**kwargs) – Keyword arguments for the write_image function

plot_grids(*remove_underlying: bool = False*)
Plots the grid layers within the builder.

Merges all of the layers together, and plots it as a single plot trace.

plot_hexbin()
Plots the hexbin layer within the builder.

If qualitative, the layer is split into uniquely labelled plot traces.

plot_outlines(*raise_errors: bool = False*)
Plots the outline layers within the builder.

All of the outlines are treated as separate plot traces. The layers must first be converted into point-like geometries.

Parameters **raise_errors** (*bool*) – Whether or not to throw errors upon reaching empty dataframes

property plot_output_service: str
Retrieves the current plot output service for the builder.

Returns The current plot output service

Return type str

plot_points()
Plots the point layers within the builder.

All of the point are treated as separate plot traces.

plot_regions()
Plots the region layers within the builder.

All of the regions are treated as separate plot traces.

remove_empties(*empty_symbol: Any = 0*, *add_to_plot: bool = True*)
Removes empty entries from the hexbin layer.

The empty entries may then be added to the plot as a grid.

Parameters

- **empty_symbol** (*Any*) – The symbol that constitutes an empty value in the layer
- **add_to_plot** (*bool*) – Whether to add the empty cells to the plot or not

remove_grid(*name: str*, *pop: bool = False*) → Dict[str, Any]
Removes a grid layer from the builder.

Parameters

- **name** – The name of the layer to remove

- **pop** (*bool*) – Whether to return the removed layer or not

Returns The removed layer (pop=True)

Return type StrDict

remove_hexbin(*pop: bool = False*) → Dict[str, Any]
Removes the main layer.

Parameters **pop** (*bool*) – Whether or not to return the removed layer

Returns The removed layer (pop=True)

Return type StrDict

remove_outline(*name: str*, *pop: bool = False*) → Dict[str, Any]
Removes an outline layer from the builder.

Parameters

- **name** – The name of the layer to remove
- **pop** (*bool*) – Whether to return the removed layer or not

Returns The removed layer (pop=True)

Return type StrDict

remove_point(*name: str*, *pop: bool = False*) → Dict[str, Any]
Removes a point layer from the builder.

Parameters

- **name** (*str*) – The name of the layer to remove
- **pop** (*bool*) – Whether to return the removed layer or not

Returns The removed layer (pop=True)

Return type StrDict

remove_region(*name: str*, *pop: bool = False*) → Dict[str, Any]
Removes a region layer from the builder.

Parameters

- **name** (*str*) – The name of the layer to remove
- **pop** (*bool*) – Whether to return the removed layer or not

Returns The removed layer (pop=True)

Return type StrDict

reset()
Resets the builder to it's initial state.

reset_data()
Resets the layers of the builder to their original state.

reset_grid_data(*name: Optional[str] = None*)
Resets the data within the grid layer to the data that was input at the beginning.

If the given name is None, all grid layers will be reset.

Parameters **name** (*str*) – The name of the layer to reset

reset_grids()
Resets the grid layer container to it's original state.

reset_hexbin_data()
Resets the data within the hexbin layer to the data that was input at the beginning.

reset_outline_data(*name: Optional[str] = None*)
Resets the data within the outline layer to the data that was input at the beginning.

If the given name is None, all outline layers will be reset.

Parameters **name** (*str*) – The name of the layer to reset

reset_outlines()
Resets the outlines within the builder to empty.

reset_point_data(*name: Optional[str] = None*)
Resets the data within the point layer to the data that was input at the beginning.

If the given name is None, all point layers will be reset.

Parameters **name** (*str*) – The name of the layer to reset

reset_points()
Resets the point layer container to its original state.

reset_region_data(*name: Optional[str] = None*)
Resets the data within the region layer to the data that was input at the beginning.

If the given name is None, all region layers will be reset.

Parameters **name** (*str*) – The name of the layer to reset

reset_regions()
Resets the regions within the builder to empty.

search(*query: str*) → Dict[str, Any]
Query the builder for specific layer(s).

Each query argument should be formatted like: <regions|grids|outlines|points|main|hexbin|all> OR <region|grid|outline|point>:<name>

And each query argument can be separated by the '+' character.

External version.

Parameters **query** (*str*) – The identifiers for the layers being searched for

Returns The result of the query

Return type StrDict

set_hexbin(*data: Union[str, pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]*, *latitude_field: Optional[str] = None*, *longitude_field: Optional[str] = None*, *hex_resolution: Optional[int] = None*, *hexbin_info: Optional[Dict[str, Any]] = None*, *manager: Optional[Dict[str, Any]] = None*)
Sets the hexbin layer to plot.

Parameters

- **data** (*DFType*) – The data for this set
- **latitude_field** (*str*) – The latitude column of the data
- **longitude_field** (*str*) – The longitude column of the data
- **hex_resolution** (*int*) – The hex resolution to use (this can also be passed via hexbin_info)
- **hexbin_info** (*StrDict*) – A container for properties pertaining to hexagonal binning

- **manager** (*StrDict*) – A container for the plotly properties for this layer

set_mapbox(*accesstoken: str*)
Prepares the builder for a mapbox output.

Sets figure.layout.mapbox_accesstoken, and plot_settings output service.

Parameters **accesstoken** (*str*) – A mapbox access token for the plot

set_plot_output_service(*service: str*)
Sets the plot output service for this builder.

Parameters **service** (*str*) – The output service (one of 'plotly', 'mapbox')

simple_clip(*method: str = 'sjoin'*)
Quick general clipping.

This function clips the hexbin layer and grid layers to the region and outline layers. The function also clips the point layers to the hexbin, region, grid, and outline layers.

Parameters **method** (*str*) – The method to use when clipping, one of 'sjoin' or 'gpd'

update_figure(*updates: Optional[Dict[str, Any]] = None*, *overwrite: bool = False*, ***kwargs*)
Updates the figure properties on the spot.

Parameters

- **updates** (*StrDict*) – A dict of properties to update the figure with
- **overwrite** (*bool*) – Whether to overwrite existing figure properties or not
- **kwargs** (**kwargs) – Any other updates for the figure

update_grid_manager(*updates: Optional[Dict[str, Any]] = None*, *overwrite: bool = False*, ***kwargs*)
Updates the general grid manager.

Parameters

- **updates** (*StrDict*) – A dict of updates for the manager
- **overwrite** (*bool*) – Whether or not to override existing manager properties
- **kwargs** (**kwargs) – Any additional updates for the manager

update_hexbin_manager(*updates: Optional[Dict[str, Any]] = None*, *overwrite: bool = False*, ***kwargs*)
Updates the manager the hexbin layer.

Parameters

- **updates** (*StrDict*) – A dict containing updates for the layer(s)
- **overwrite** (*bool*) – Whether to override the current properties with the new ones or not
- **kwargs** (**kwargs) – Other updates for the layer(s)

update_outline_manager(*name: Optional[str] = None*, *updates: Optional[Dict[str, Any]] = None*, *overwrite: bool = False*, ***kwargs*)
Updates the manager of a outline or outlines.

The manager consists of Plotly properties. If the given name is none, all outline layers will be updated.

Parameters

- **name** (*str*) – The name of the layer to update
- **updates** (*StrDict*) – A dict containing updates for the layer(s)
- **overwrite** (*bool*) – Whether to override the current properties with the new ones or not

- **kwargs** (**kwargs) – Other updates for the layer(s)

update_point_manager(*name: Optional[str] = None*, *updates: Optional[Dict[str, Any]] = None*, *overwrite: bool = False*, ***kwargs*)

Updates the manager of a point or points.

The manager consists of Plotly properties. If the given name is none, all point layers will be updated.

Parameters

- **name** (*str*) – The name of the layer to update
- **updates** (*StrDict*) – A dict containing updates for the layer(s)
- **overwrite** (*bool*) – Whether to override the current properties with the new ones or not
- **kwargs** (**kwargs) – Other updates for the layer(s)

update_region_manager(*name: Optional[str] = None*, *updates: Optional[Dict[str, Any]] = None*, *overwrite: bool = False*, ***kwargs*)

Updates the manager of a region or regions.

The manager consists of Plotly properties. If the given name is none, all region layers will be updated.

Parameters

- **name** (*str*) – The name of the layer to update
- **updates** (*StrDict*) – A dict containing updates for the layer(s)
- **overwrite** (*bool*) – Whether to override the current properties with the new ones or not
- **kwargs** (**kwargs) – Other updates for the layer(s)

class geohexviz.builder.**PlotStatus**(*value*)

An enumeration of different plot status.

CHAPTER

TWO

GEOHEXVIZ - ERRORS MODULE

exception geohexviz.errors.**BigQueryError**(*problematic: str*)

exception geohexviz.errors.**BinValueTypeError**(*message: str = 'The data can not be empty.'*)

exception geohexviz.errors.**BuilderAlterationError**(*message: str = 'There was an error while altering data within the builder.'*)

exception geohexviz.errors.**BuilderPlotBuildError**(*message: str = 'An error occurred while plotting.'*)

exception geohexviz.errors.**BuilderQueryInvalidError**(*message: str = 'The input query was invalid.'*)

exception geohexviz.errors.**ColorScaleError**(*message: str = 'There was an error while reading the colorscale.'*)

exception geohexviz.errors.**DataEmptyError**(*message: str = 'The data can not be empty.'*)

exception geohexviz.errors.**DataFileReadError**(*message: str = 'There was an error while reading the data file.'*)

exception geohexviz.errors.**DataReadError**(*name: str*, *dstype:* geohexviz.errors.LayerType, *allow_builtin: bool*)

exception geohexviz.errors.**GeometryParseLatLongError**(*name: str*, *dstype:* geohexviz.errors.LayerType, *lat: bool*)

exception geohexviz.errors.**LatLongParseTypeError**(*name: str*, *dstype:* geohexviz.errors.LayerType, *lat: bool*)

exception geohexviz.errors.**LayerNamingError**(*name: str*, *dstype:* geohexviz.errors.LayerType, *err_type: str*)

class geohexviz.errors.**LayerType**(*value*)

An enumeration of different dataset types.

exception geohexviz.errors.**NoFilepathError**(*message: str = 'There was no filepath provided.'*)

exception geohexviz.errors.**NoHexagonalTilingError**(*name: str*, *dstype:* geohexviz.errors.LayerType)

exception geohexviz.errors.**NoLayerError**(*name: str*, *dstype:* geohexviz.errors.LayerType)

exception geohexviz.errors.**NoLayersError**(*dstype:* geohexviz.errors.LayerType)

CHAPTER

THREE

GEOHEXVIZ - TEMPLATES MODULE

geohexviz.templates.**get_template**(*name: str*) → dict
Retrieves a template from the module.

Parameters **name** (*str*) – The name of the template

Returns The retrieved template

Return type dict

CHAPTER

FOUR

GEOHEXSIMPLE - SIMPLE MODULE

scripts.geohexsimple.simple.**run_json**(*filepath: str*, *strict: bool = False*, *debug: bool = False*)
Runs a json file representation of a plot scheme.

Parameters

- **filepath** (*str*) – The filepath to the json file
- **debug** (*bool*) – Determines if parts of the internal processes are printed or not
- **strict** (*bool*) – Whether to print error messages during function applications or not

CHAPTER

FIVE

FURTHER INFORMATION

GeoHexViz is a package for the simple and repeatable visualization of hexagon-ally binned data sets. The package's main feature is a PlotBuilder class which utilizes tools to hexagon-ally bin your dataset and then display it.

5.1 Command Line / JSON Usage

The GeoHexViz distribution includes a module that can allow the reading of JSON files for quick and easy plots.

```
{
  "main_dataset": {
    "data": "<sample csv file>",
    "hex_resolution": 4
  },
  "output_figure": {
    "filepath": "<sample filepath>",
    "width": 600,
    "height": 400
  },
  "display_figure": true
}
```

running the JSON script will allow you to input a JSON file via command line.

You may also use the functions that allow the executing of a JSON file from a python module.

```
from scripts.geohexsimple.simple import run_json

run_json("<filepath here>")
```

5.2 Python Module Usage (PlotBuilder)

```
from pandas import DataFrame
from geohexviz.builder import PlotBuilder

# Creating an example dataset
inputdf = DataFrame(dict(
    latitude=[17.57, 17.57, 17.57, 19.98, 19.98, 46.75],
    longitude=[10.11, 10.11, 10.12, 50.55, 50.55, 31.17],
```

(continues on next page)

(continued from previous page)

```
    value=[120, 120, 120, 400, 400, 700]
))

# Instantiating builder
builder = PlotBuilder()
builder.set_hexbin(inputdf, hexbin_info=dict(binning_fn='sum', binning_field='value'))

builder.finalize(raise_errors=False)
builder.display(clear_figure=True)

# A mapbox map
builder.set_mapbox('<ACCESS TOKEN>')
builder.finalize()
builder.display(clear_figure=True)
```

5.2.1 Behind the Scenes

When the main dataset is passed into the builder, the data is processed in the following steps:

Data:

index	lats	lons	value
0	17.57	10.11	120
1	17.57	10.11	120
2	17.57	10.12	120
3	19.98	50.55	400
4	19.98	50.55	400
5	46.75	31.17	700

1) Coordinate columns are converted into geometry (if applicable)

index	value	geometry
0	120	POINT(17.57, 10.11)
1	120	POINT(17.57, 10.11)
2	120	POINT(17.57, 10.12)
3	400	POINT(19.98, 50.55)
4	400	POINT(19.98, 50.55)
5	700	POINT(46.75, 31.17)

2) Hex cells are then placed over the data

hex	value	geometry
83595affffffff	120	POINT(17.57, 10.11)
83595affffffff	120	POINT(17.57, 10.11)
83595affffffff	120	POINT(17.57, 10.11)
835262ffffffff	400	POINT(19.98, 50.55)
835262ffffffff	400	POINT(19.98, 50.55)
831e5dffffffff	700	POINT(46.75, 31.17)

(hex resolution = 3)

3) The data is grouped together by hex, and hex geometry is added

hex	items	value_field	geometry
83595affffffff	(120,120,120)	360	POLYGON ((30.57051 46.80615, 30.47843 46.19931…
835262ffffffff	(400, 400)	800	POLYGON ((49.90903 20.19437, 49.74835 19.60088…
831e5dffffffff	(700)	700	POLYGON ((9.44614 17.39197, 9.49704 16.75205, …

(binning function = sum of grouped values)

When the data is eventually plotted, a GeoJSON format of the data is passed alongside plotly properties are passed to the Plotly graphing library.

5.3 Installation

As of right now the GeoHexViz package can be cloned on GitHub, and install by using the `setup.py` file.

In the near future the package will be available via pip: Use the package manager pip to install GeoHexViz.

```
pip install geohexviz
```

5.4 Further Documentation

There is further documentation contained within the Reference Document published alongside this software package, which is available {HERE}. The official API documentation is also available {HERE}.

5.5 Acknowledgements

Thank you to Nicholi Shiell for his input in testing, and providing advice for the development of this package.

5.6 Limitations

This package uses GeoJSON format to plot data sets. With GeoJSON comes difficulties when geometries cross the 180th meridian . The issue appears to cause a color that bleeds through the entire plot and leaves a hexagon empty. In the final plot, this issue may or may not appear as it only occurs at certain angles of rotation. In this package a simple solution to the problem is implemented, in the future it would be best to provide a more robust solution. The solution that is used works generally, however, when hexagons containing either the north or south pole are present, the solution to the 180th meridian issue persists. This pole issue can be seen below.

There also exists some issues with the generation of discrete color scales under rare circumstances. These circumstances include generating discrete color scales with not enough hues to fill the scale, and generating diverging discrete colorscales with the center hue in a weird position. These issues have been noted and will be fixed in the near future.

There exists issues with the positioning and height of the color bar with respect to the plot area of the figure. Although the user is capable of altering the dimensions and positioning of the color bar, this should be done automatically as it is a common feature of publication quality choropleth maps.

5.7 Contributing

For major changes, please open an issue first to discuss what you would like to change.

5.8 Contact

For any questions, feedback, bug reports, feature requests, etc please first present your thoughts via GitHub issues. For further assistance please contact tonyabouzeidan@cmail.carleton.ca

5.9 Copyright and License

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

s

INDEX

L

M

N

O

P

R

S

U