GeoHexViz

Release 1.0.0

Tony Abou Zeidan

Nov 05, 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)*)

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

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 UTILS PACKAGE

The utils package contains all of the utility functions used by GeoHexViz.

2.1 File Util Module

This module contains functions for the running of plot properties files. This module is used by the GeoHexSimple script (geohexviz/simple.py).

geohexviz.utils.file.**run_file**(*filepath: str*, *strict: bool = False*, *debug: bool = False*)
Runs a file that represents plot properties.

Accepted: YAML, JSON

Parameters

- **filepath** (*str*) – The filepath containing the plot properties
- **strict** (*bool*) – Whether to handle strict events or not
- **debug** (*bool*) – Whether to handle informational events or not

Returns The status of the final plot

Return type *PlotStatus*

geohexviz.utils.file.**run_json**(*filepath: str*, *strict: bool = False*, *debug: bool = False*)
Runs a JSON representation of plot properties.

Parameters

- **filepath** (*str*) – The path to the JSON file
- **strict** (*bool*) – Whether to handle strict events or not
- **debug** (*bool*) – Whether to handle informational events or not

Returns The status of the final plot

Return type *PlotStatus*

geohexviz.utils.file.**run_yaml**(*filepath: str*, *strict: bool = False*, *debug: bool = False*)
Runs a YAML representation of plot properties.

Parameters

- **filepath** (*str*) – The path to the YAML file
- **strict** (*bool*) – Whether to handle strict events or not
- **debug** (*bool*) – Whether to handle informational events or not

Returns The status of the final plot

Return type *PlotStatus*

2.2 Geo Util Module

This module contains all of the functions that GeoHexViz utilizes to do geospatial processing.

geohexviz.utils.geoutils.**add_geometry**(*row*) → shapely.geometry.polygon.Polygon

Returns a Polygon of the hex id in the same row.

This function returns a Polygon representing the boundaries defined by the hex cell in the given row.

Parameters

- **row** – A row in the dataframe
- **hex_field_id** –

Returns A polygon built from a hex id

Return type Polygon

geohexviz.utils.geoutils.**apply_bin_function**(*hex_gdf: pandas.core.frame.DataFrame*, *binning_field: str*, *binning_fn: Callable*, *binning_args=None*, *result_name: Optional[str] = None*, ***binning_kw*)

Applies a function to a grouped dataframe (intended for hex use).

Parameters

- **hex_gdf** (*DataFrame*) – The dataframe to perform the function on
- **binning_field** (*str*) – The column within the dataframe to apply the function on
- **binning_fn** (*Callable*) – The function to apply
- **binning_args** (*Iterable*) – Arguments for the function
- **result_name** (*str*) – The name of the column that contains the result
- **binning_kw** (**kwargs) – Keyword arguments for the function

geohexviz.utils.geoutils.**bin_by_hexid**(*hex_gdf: Union[pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]*, *binning_field: Optional[str] = None*, *binning_fn: Optional[Callable] = None*, *binning_args=None*, *hex_field: Optional[str] = None*, *result_name: str = 'value_field'*, *add_geoms: bool = False*, *loss_method: bool = True*, ***binning_kw*)

Bins a DataFrame by hex cell ids.

This function assumes the dataframe has a VALID hex and geometry columns. Using these columns the data is grouped into what hexagon on the grid they fall into.

Parameters

- **hex_gdf** (*GeoDataFrame*) – A dataframe representing any kind of hex data
- **binning_field** (*str*) – The column that the data will be grouped by
- **binning_fn** (*Callable*) – The function to perform after grouping
- **binning_args** (*args) – Arguments for the binning function
- **hex_field** (*str*) – The location of the hex ids in the dataframe (None->index)

- **result_name** (*str*) – The name of the column that contains the grouped result after having the function applied
- **add_geoms** (*bool*) – Whether to add hex geometries after grouping or not
- **loss_method** (*bool*) – Whether or not to use a method that is quicker but provides a loss of data (columns)

Returns A frame containing the binned hex grid and its geometries

Return type GeoDataFrame

geohexviz.utils.geoutils.**check_crossing**(*lon1: float*, *lon2: float*, *validate: bool = True*)

Assuming a minimum travel distance between two provided longitude coordinates, checks if the 180th meridian (antimeridian) is crossed.

geohexviz.utils.geoutils.**conform_geogeometry**(*gdf: geopandas.geodataframe.GeoDataFrame*, *d3_geo: bool = True*, *fix_polys: bool = True*) → geopandas.geodataframe.GeoDataFrame

Fixes the winding order and antimeridian crossings for geometries in a GeoDataFrame.

Parameters

- **gdf** (*GeoDataFrame*) – The geodataframe to conform
- **d3_geo** (*bool*) – Whether to orient the polygons clockwise or counter-clockwise
- **fix_polys** (*bool*) – Whether to fix antimeridian crossings or not

Returns The conformed geodataframe

Return type GeoDataFrame

geohexviz.utils.geoutils.**conform_polygon**(*poly: Union[shapely.geometry.polygon.Polygon, shapely.geometry.multipolygon.MultiPolygon]*, *rfc7946: bool = True*, *fix_poly: bool = True*) → Union[shapely.geometry.polygon.Polygon, shapely.geometry.multipolygon.MultiPolygon]

Conforms the given polygon to the given standard.

Parameters

- **poly** (*Polygon*) – The polygon to conform
- **rfc7946** (*bool*) – The conform standard (True->Clockwise,False->Counterclockwise)
- **fix_poly** (*bool*) – Whether or not to fix the polygons if they cross the anti-meridian (by shifting)

Returns The conformed Polygon

Return type Polygon

geohexviz.utils.geoutils.**convert_crs**(*left: geopandas.geodataframe.GeoDataFrame*, *right: geopandas.geodataframe.GeoDataFrame*, *crs: Any = 'EPSG:4326'*)

Converts two GeoDataFrames to the same crs.

Parameters

- **left** (*GeoDataFrame*) – The left dataframe
- **right** (*GeoDataFrame*) – The right dataframe
- **crs** (*Any*) – The crs that the two dataframes will be converted to

geohexviz.utils.geoutils.**convert_dataframe_geometry_to_geodataframe**(*df: pandas.core.frame.DataFrame*, *geometry_field: str = 'geometry'*) → geopandas.geodataframe.GeoDataFrame

Converts the given dataframe into a GeoDataFrame based on pre-existing geometry.

This function converts the given dataframe into a GeoDataFrame based on a pre-existing geometry column in the dataframe.

Parameters

- **df** (*DataFrame*) – Any dataframe with a pre-existing geometry column
- **geometry_field** (*str*) – The column that contains geometry

Returns A geodataframe of the given dataframe

Return type GeoDataFrame

geohexviz.utils.geoutils.**find_geoms_within_collection**(*gc*, *collapse: bool = False*) → set

Finds the geometry types within a collection of geometries.

Parameters

- **gc** – The collection of geometries
- **collapse** (*bool*) – Whether or not to collapse Multi geometries into their sub counterparts

Returns The geometry types found

Return type set

geohexviz.utils.geoutils.**generate_grid_over**(*gdf: geopandas.geodataframe.GeoDataFrame*, *hex_resolution: int*) → geopandas.geodataframe.GeoDataFrame

This function generates a hexagonal grid around a dataframe (a box).

Parameters

- **gdf** (*GeoDataFrame*) – The dataframe to generate a hex-box around
- **hex_resolution** (*int*) – The resolution to use for the grid

Returns The resulting grid box

Return type GeoDataFrame

geohexviz.utils.geoutils.**get_present_geomtypes**(*gdf: geopandas.geodataframe.GeoDataFrame*, *allow_collections: bool = True*, *collapse_geoms: bool = False*) → set

Obtains a set of unique geometry types within a geodataframe.

Parameters

- **gdf** (*GeoDataFrame*) – The geodataframe to find geometry types of
- **allow_collections** (*bool*) – Whether or not to parse GeometryCollections for their types
- **collapse_geoms** (*bool*) – Whether or not to collapse Multi geometries into their sub geometries

Returns The unique set of geometries within the geodataframe

Return type set

geohexviz.utils.geoutils.**gpdclip**(*clip: geopandas.geodataframe.GeoDataFrame*, *to: geopandas.geodataframe.GeoDataFrame*, *enforce_crs: Any = 'EPSG:4326'*, *keep_geom_type: bool = True*)

Clips a GeoDataFrame to another via GeoPandas clip function.

The operation first converts the two dataframes to the same crs.

Parameters

- **clip** (*GeoDataFrame*) – The dataframe that is being clipped to the other dataframes boundary
- **to** (*GeoDataFrame*) – The dataframe that is acting like the boundary for the clipping
- **enforce_crs** (*Any*) – The crs to enforce before clipping
- **keep_geom_type** (*bool*) – Passed into geopandas clip function

Returns The clipped dataframe

Return type GeoDataFrame

geohexviz.utils.geoutils.**hexify_dataframe**(*gdf: geopandas.geodataframe.GeoDataFrame*, *hex_resolution: int*, *add_geom: bool = False*, *keep_geom: bool = False*, *old_geom_name: Optional[str] = None*, *as_index: bool = True*, *raise_errors: bool = False*) → geopandas.geodataframe.GeoDataFrame

Makes a new GeoDataFrame, with the index set as the hex cell ids that each geometry in the geometry column corresponds to.

Parameters

- **gdf** (*GeoDataFrame*) – The GeoDataFrame to place a hex cell id overlay on
- **hex_resolution** (*int*) – The resolution of the hexes to be generated
- **add_geom** (*bool*) – Whether to add the hex geometry to the dataframe or not
- **keep_geom** (*bool*) – Whether to keep old geometry or not (add_geom=True)
- **old_geom_name** (*str*) – The name of the column to store the old geometry in (add_geom=True, keep_geom=True)
- **as_index** (*bool*) – Whether to make the hex column the index or not
- **raise_errors** (*bool*) – Whether to raise errors related to empty geometry or not

Returns A GeoDataFrame with a hex id index

Return type GeoDataFrame

geohexviz.utils.geoutils.**hexify_geometry**(*gdf: Union[pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]*, *hex_col: Optional[str] = None*, *keep_geoms: bool = False*, *old_geom_name: Optional[str] = None*) → geopandas.geodataframe.GeoDataFrame

Adds the geometry of the hex ids in the given column or index.

Parameters

- **gdf** (*GeoDataFrame*) – The GeoDataFrame containing the hex ids
- **hex_col** (*str*) – The column containing hex ids (None->index)
- **keep_geoms** (*bool*) – Whether or not to keep old geometry

- **old_geom_name** (*str*) – The name for the column containing old geometry (keep_geoms=True)

Returns A GeoDataFrame with hex ids and their geometries

Return type GeoDataFrame

geohexviz.utils.geoutils.**merge_datasets_simple**(**args*, *merge_op: Callable = <built-in function add>*, *common_columns: Optional[List[str]] = None*, *keep_columns: Optional[List[str]] = None*, *drop: bool = True*, *crs: Optional[str] = None*, *result_name: Optional[str] = None*, *errors: str = 'ignore'*) → geopandas.geodataframe.GeoDataFrame

Merges the datasets with the given merge operation.

Parameters

- **args** (*args: List[Union[GeoDataFrame, Tuple[str, GeoDataFrame]]]) – The datasets to merge
- **merge_op** (*Callable*) – The merge operation to perform on the datasets
- **keep_columns** (*List[str]*) – Additional columns to keep
- **common_columns** (*List[str]*) – Columns that ALL dataframes share in common
- **drop** (*bool*) – Whether to drop unnecessary columns or not
- **crs** (*Optional[str]*) – The crs of the new dataframe
- **result_name** (*Optional[str]*) – The name of the resulting column within the dataframe
- **errors** (*str*) – The parameter determining errors to be thrown

Returns A merged dataframe

Return type GeoDataFrame

geohexviz.utils.geoutils.**pointify_geodataframe**(*gdf: geopandas.geodataframe.GeoDataFrame*, *keep_geoms: bool = True*, *raise_errors: bool = True*) → geopandas.geodataframe.GeoDataFrame

Makes a new GeoDataFrame, with the geometry all being of Point type, and the index representing the original row in the dataframe.

Parameters

- **gdf** (*GeoDataFrame*) – The GeoDataFrame to convert the geometries of
- **keep_geoms** (*bool*) – Whether to keep the original geometries in the dataframe or not
- **raise_errors** (*bool*) – Errors are raised by pandas when the dataframe has no geometry, throw or not

Returns A GeoDataFrame with a point-only geometry

Return type GeoDataFrame

geohexviz.utils.geoutils.**remove_other_geometries**(*gdf: geopandas.geodataframe.GeoDataFrame*, *geom_type: str*) → geopandas.geodataframe.GeoDataFrame

Removes unwanted geometry from the GeoDataFrame.

Parameters

- **gdf** (*GeoDataFrame*) – The GeoDataFrame to remove from

- **geom_type** (*str*) – The geometry type that is to be kept

Returns The GeoDataFrame without any other geometries

Return type GeoDataFrame

geohexviz.utils.geoutils.**simple_geojson**(*gdf: geopandas.geodataframe.GeoDataFrame*, *value_field: Optional[str] = None*, *id_field: Optional[str] = None*, *file_output: Optional[str] = None*) → geojson.feature.FeatureCollection

Provides a GeoJSON representation of a GeoDataFrame.

Only works on GeoDataFrames with id, value, and geometry columns.

Parameters

- **gdf** (*GeoDataFrame*) – The geodataframe to make a GeoJSON of
- **value_field** (*str*) – The value column of the dataframe
- **id_field** (*Optional[str]*) – An id field to use for the GeoJSON
- **file_output** (*Optional[str]*) – Filepath for file output

Returns A GeoJSON-like representation of the geodataframe

Return type FeatureCollection

geohexviz.utils.geoutils.**sjoinclip**(*clip: geopandas.geodataframe.GeoDataFrame*, *to: geopandas.geodataframe.GeoDataFrame*, *operation: str = 'intersects'*, *enforce_crs: Any = 'EPSG:4326'*) → geopandas.geodataframe.GeoDataFrame

Clips a GeoDataFrame to another via GeoPandas spatial join operations.

The operation first converts the two GeoDataFrames to the same crs.

Parameters

- **clip** (*GeoDataFrame*) – The dataframe that is being clipped to the other dataframes boundary
- **to** (*GeoDataFrame*) – The dataframe that is acting like the boundary for the clipping
- **operation** (*str*) – The operation to be performed in the spatial join
- **enforce_crs** (*Any*) – The crs to enforce before clipping

Returns The clipped dataframe

Return type GeoDataFrame

geohexviz.utils.geoutils.**unify_geodataframe**(*gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Unifies the geometries in a GeoDataFrame into a new GeoDataFrame.

Parameters **gdf** (*GeoDataFrame*) – The input dataframe

2.3 Plot Util Module

This module contains all of the functions that GeoHexViz uses to modify the properties of its internal plot and layers.

geohexviz.utils.plot_util.**format_html_exp10**(*exp: float*, *exp_type: str*) → str
Formats a exponent to the power of 10 in html form.

geohexviz.utils.plot_util.**format_latex_exp10**(*exp: float*, *exp_type: str*) → str
Formats a exponent to the power of 10 in latex form.

geohexviz.utils.plot_util.**format_raw_exp10**(*exp: float*, *exp_type: str*) → str
Formats a exponent to the power of 10 in raw form.

geohexviz.utils.plot_util.**get_shapes_from_world**(*name: Optional[str] = None*) → geopandas.geodataframe.GeoDataFrame
Retrieves a continent or country from the world, or the whole world.

This function uses GeoPandas naturalearth_lowres dataset. If the given name is None, the function will retrieve the shapes of thw entire world.

Parameters **name** (*str*) – The name of the continent or country to get the shapes for

Returns The dataframe containing the geometry of the country, continent, or world

Return type GeoDataFrame

geohexviz.utils.plot_util.**gpd_clip**(*clip: geopandas.geodataframe.GeoDataFrame*, *to: List[geopandas.geodataframe.GeoDataFrame]*, *validate: bool = True*)
Plot wrapper for the geopandas clip operation.

Parameters

- **clip** (*GeoDataFrame*) – The data to clip
- **to** (*List[GeoDataFrame]*) – The list of data to act as the clip
- **validate** (*bool*) – Whether or not to validate the result or not (throws)

Returns The result of the clip

Return type GeoDataFrame

geohexviz.utils.plot_util.**logify_info**(*values: Union[Sequence[float], Set[float]]*, *text_type: str = 'raw'*, *exp_type: Optional[str] = None*, *fill_last: bool = True*, *include_min: bool = False*, *include_max: bool = False*, *minmax_rounding: int = 3*, *include_predecessors: bool = False*, *min_prefix: str = ''*, *min_suffix: str = ''*, *max_prefix: str = ''*, *max_suffix: str = ''*) → Dict[str, Any]
Retrieves a dictionary of information for a log scale.

entries: scale-min -> The minimum value on the scale scale-max -> The maximum value on the scale original-values -> The values given logged-values -> The values after having log10 performed on them scale-dict -> The scale values and their text

Parameters

- **values** (*Union[Sequence[float], Set[float]]*) – The values to compute a log scale for
- **text_type** (*Optional[str]*) – Determines how to format the tick text, (latex, html, raw)
- **exp_type** (*Optional[str]*) – What type of exponent to select (None, E, ^, *, r)

- **fill_last** (*bool*) – Whether to extend the last segment of the log scale to the next highest power or not
- **include_min** (*bool*) – Whether to include a value-text pair for the minimum value
- **include_max** (*bool*) – Whether to include a value-text pair for the maximum value
- **minmax_rounding** (*int*) – The number of decimals to round the min and max values to
- **include_predecessors** (*bool*) – Whether to include all previous exponents in the scale values or not
- **min_prefix** (*str*) – Prefix for the minimum value-text pair
- **min_suffix** (*str*) – Suffix for the minimum value-text pair
- **max_prefix** (*str*) – Prefix for the maximum value-text pair
- **max_suffix** (*str*) – Suffix for the maximum value-text pair

Returns A dictionary of the above information

Return type dict

geohexviz.utils.plot_util.**logify_info_dep**(*values: Union[Sequence[float], Set[float]]*, *text_type: str = 'raw'*, *exp_type: Optional[str] = None*, *fill_first: bool = True*, *fill_last: bool = True*, *include_min: bool = False*, *include_max: bool = False*, *minmax_rounding: int = 6*, *min_prefix: str = ''*, *min_suffix: str = ''*, *max_prefix: str = ''*, *max_suffix: str = ''*) → Dict[str, Any]

Retrieves a dictionary of information for a log scale.

DEPRECATED

entries: scale-min -> The minimum value on the scale scale-max -> The maximum value on the scale original-values -> The values given logged-values -> The values after having log10 performed on them scale-vals -> The scale values scale-text -> The text for each scale values

Parameters

- **values** (*Union[Sequence[float], Set[float]]*) – The values to compute a log scale for
- **text_type** (*Optional[str]*) – Determines how to format the tick text, (latex, html, raw)
- **exp_type** (*Optional[str]*) – What type of exponent to select (None, E, ^, *, r)
- **fill_first** (*bool*) – Whether to extend the first segment of the log scale to the next lowest power or not
- **fill_last** (*bool*) – Whether to extend the last segment of the log scale to the next highest power or not
- **include_min** (*bool*) – Whether to include a value-text pair for the minimum value
- **include_max** (*bool*) – Whether to include a value-text pair for the maximum value
- **minmax_rounding** (*int*) – The number of decimals to round the min and max values to
- **min_prefix** (*str*) – Prefix for the minimum value-text pair
- **min_suffix** (*str*) – Suffix for the minimum value-text pair
- **max_prefix** (*str*) – Prefix for the maximum value-text pair
- **max_suffix** (*str*) – Suffix for the maximum value-text pair

Returns A dictionary of the above information

Return type dict

geohexviz.utils.plot_util.**logify_scale**(*df: pandas.core.frame.DataFrame*, ***kwargs*) → Dict[str, Any]
Converts a manager into log scale form.

Parameters

- **df** (*DataFrame*) – The dataframe that contains the values for the scale
- **kwargs** (**kwargs) – Keyword arguments to be passed into logify_info

geohexviz.utils.plot_util.**logify_scale_dep**(*df: pandas.core.frame.DataFrame*, ***kwargs*) → Dict[str, Any]
Converts a manager into log scale form.

DEPRECATED

Parameters

- **df** (*DataFrame*) – The dataframe that contains the values for the scale
- **kwargs** (**kwargs) – Keyword arguments to be passed into logify_info

geohexviz.utils.plot_util.**opacify_colorscale**(*dataset: dict*, *alpha: Optional[float] = None*)
Plot wrapper for adjusting the opacity of a colorscale.

Parameters

- **dataset** (*dict*) – The dataset whose colorscale to adjust
- **alpha** (*float*) – The alpha to adjust by (if not, then the alpha within the dataset is used)

geohexviz.utils.plot_util.**sjoin_clip**(*clip: geopandas.geodataframe.GeoDataFrame*, *to: List[geopandas.geodataframe.GeoDataFrame]*, *operation: str = 'intersects'*, *validate: bool = False*) → geopandas.geodataframe.GeoDataFrame
Plot wrapper for the spatial join clip operation.

Parameters

- **clip** (*GeoDataFrame*) – The data to clip
- **to** (*List[GeoDataFrame]*) – The list of data to act as the clip
- **operation** (*str*) – The operation to be performed in the spatial join
- **validate** (*bool*) – Whether or not to validate the result or not (throws)

Returns The result of the clip

Return type GeoDataFrame

2.4 General Util Module

This module contains other utility functions used by GeoHexViz.

geohexviz.utils.util.**dict_deep_update**(*d: dict*, *u: dict*) → object
Updates a dict without changing nested dicts that may be present.

Parameters

- **d** (*dict*) – The dict to update

- **u** (*dict*) – The updating dict

geohexviz.utils.util.**fix_filepath**(*filepath: str*, *add_filename: str = ''*, *add_ext: str = ''*) → str
Converts a directorypath, or filepath into a valid filepath.

Parameters

- **filepath** (*str*) – The filepath to convert
- **add_filename** (*str*) – The filename to add if there is none
- **add_ext** (*str*) – The extension to add if there is a filename

Returns The converted filepath

Return type str

geohexviz.utils.util.**get_best**(**args*, ***kwargs*) → object
Gets the best option from a list of labels.

see get_sorted_best()

Parameters

- **args** (*args) – Arguments to be passed into the get_sorted_best() function
- **kwargs** (**kwargs) – Keywords to be passed into the get_sorted_best() function

Returns The best option in the list

Return type object

geohexviz.utils.util.**get_occurrences**(*lst: list*, ***kwargs*) → list
Retrieves a list of tuples containing the list item and its frequency in the list.

Parameters

- **lst** (*list*) – The list to count frequencies from
- **kwargs** (**kwargs) – Keyword arguments for the sorted function

Returns The list of item frequency pairs

Return type list

geohexviz.utils.util.**get_sorted_best**(*lst: list*, *allow_ties: bool = True*, *join_ties: bool = True*, *selector: Optional[Iterable] = None*, *reverse: bool = True*) → object
Retrieves the best entry or entries from a list of labels based on occurrences.

Parameters

- **lst** (*list*) – The list of labels to parse the best or worst option from
- **allow_ties** (*bool*) – Whether to allow ties between labels or not
- **join_ties** (*bool*) – Whether to join the labels of the tie if present or not
- **selector** (*Iterable*) – If multiple items are tied, this determines the order of which they will be selected as the best
- **reverse** (*bool*) – Whether to reverse the list or not (reversed=best, !reversed=worst)

Returns The best or worst option from the list

Return type object

geohexviz.utils.util.**get_worst**(**args*, ***kwargs*) → object

Gets the worst option from a list of labels.

see get_sorted_best()

Parameters

- **args** (*args) – Arguments to be passed into the get_sorted_best() function
- **kwargs** (**kwargs) – Keywords to be passed into the get_sorted_best() function

Returns The worst option in the list

Return type object

2.5 Colour Scale Util Module

This module contains all of the functions that are dedicated to generating valid colour scales for Plotly.

geohexviz.utils.colorscales.**discretize**(*colors: List[str]*, *size_portion: float = 0*, *center_portion: float = 0*, *fix_bound: bool = True*, *fix_extension: bool = True*) → List[Tuple[float, str]]

Takes a single sequential colorscale and gets it's discrete form.

Parameters

- **colors** (*List[str]*) – The list of colors on the colors
- **size_portion** (*float*) – The amount of space each discrete section will occupy on the colors (decimal-percentage)
- **center_portion** (*float*) – The amount of space that the center will take up on the colors (decimal-percentage)
- **fix_bound** (*bool*) – Determines if a color is fixed if it goes over the top of the colors
- **fix_extension** (*bool*) – Determines if the last color should reach the end of the colors if no more colors available

Returns The discrete colorscale

Return type List[Tuple[float, str]]

geohexviz.utils.colorscales.**discretize2**(*colors: List[str]*, *size_portion: float = 0*, *center_portion: float = 0.0*, *size_low: float = 0.0*, *size_high: float = 1.0*, *fix_bound: bool = True*, *fix_extension: bool = True*) → geohexviz.utils.colorscales._DScale

Takes a single sequential colorscale and gets it's discrete form.

Parameters

- **colors** (*List[str]*) – The list of colors on the colors
- **size_portion** (*float*) – The amount of space each discrete section will occupy on the colors (decimal-percentage)
- **fix_bound** (*bool*) – Determines if a color is fixed if it goes over the top of the colors
- **fix_extension** (*bool*) – Determines if the last color should reach the end of the colors if no more colors available

Returns The discrete colorscale

Return type List[Tuple[float, str]]

geohexviz.utils.colorscales.**discretize_cscale**(*colorscale*, *scale_type: str*, *low: float*, *high: float*, ***kwargs*)

Transforms a normal colorscale into a discrete colorscale.

Parameters

- **colorscale** (*Any*) – The colorscale to be converted
- **scale_type** (*str*) – The type of colorscale (sequential, diverging)
- **low** (*float*) – The minimum value on the colors
- **high** (*float*) – The maximum value on the colors
- **kwargs** (**kwargs) – Keyword arguments for the discrete functions

Returns The discretized colorscale

Return type Any

geohexviz.utils.colorscales.**discretize_diverging**(*colors: List[str]*, *low: float*, *high: float*, *discrete_size: float = 1.0*, *remove_middle: bool = True*, *high_shading: bool = True*, *center: Optional[float] = None*, *center_hue: Optional[int] = None*, *choose_left_hues: Union[List[int], int] = 1*, *choose_right_hues: Union[List[int], int] = 1*, *choose_left_luminance: float = 0.0*, *choose_right_luminance: float = 0.0*, *choose_luminance: float = 0.0*) → List[Tuple[float, str]]

Transforms a diverging scale into a discrete diverging scale.

It should be noted that luminance parameters only work if the scale is in RGB format. THIS FEATURE IS HIGHLY EXPERIMENTAL.

Parameters

- **colors** (*List[str]*) – The list of colors on the scale
- **low** (*float*) – The minimum value on the scale
- **high** (*float*) – The maximum value on the scale
- **discrete_size** (*float*) – The amount of space each discrete bin takes on the colorscale
- **remove_middle** (*bool*) – Whether to remove the center of the diverging scale or not
- **high_shading** (*bool*) – Whether to use hues closer to the ends of the scale or not
- **center** (*float*) – The position of the center on the scale
- **center_hue** (*int*) – Where the center hue is in the list of colors given
- **choose_left_hues** (*Union[List[int], int]*) – The list of color positions to use on the left of the scale or an integer skip for color selection on the left
- **choose_right_hues** (*Union[List[int], int]*) – The list of color positions to use on the right of the scale or an integer skip for color selection on the right
- **choose_left_luminance** (*float*) – Choose colors for the left side under the maximum luminance
- **choose_right_luminance** (*float*) – Choose colors for the right side under the maximum luminance

- **choose_luminance** (*float*) – Choose colors for the both sides of the scale under the maximum luminance

Returns The discretized diverging colorscale

Return type Any

geohexviz.utils.colorscales.**discretize_sequential**(*colors: List[str]*, *low: float*, *high: float*, *discrete_size: float = 1.0*, *choose_hues: Union[List[int], int] = 1*, *choose_luminance: float = 0.0*) → List[Tuple[float, str]]

Makes a sequential colors discrete based on min, max on the colors.

choose_luminance does not work with colors that are not in rgb format.

Parameters

- **colors** (*List[str]*) – The list of colors on the colors
- **low** (*float*) – The minimum numerical value on the colors (not percentage)
- **high** (*float*) – The maximum numerical value on the colors (not percentage)
- **discrete_size** (*float*) – The numerical amount that each discrete bar will occupy (not percentage)
- **choose_hues** (*Union[List[int],int]*) – Determines the step used in selecting colors from the colors, or the list of color positions that are used
- **choose_luminance** (*float*) – The maximum luminance of the colors to be selected

Returns The discrete sequential colors

Return type List[Tuple[float, str]]

geohexviz.utils.colorscales.**get_cscale_format**(*colorscale*) → str

Retrieves the format of a colorscale.

Parameters **colorscale** (*Any*) – The colorscale to get the format of

Returns The format of the colorscale (string, iterable, nested iterable, unknown)

Return type str

geohexviz.utils.colorscales.**solid_scale**(*color: str*, *min_scale: float = 0.0*, *max_scale: float = 1.0*) → Tuple

Retrieves a solid colorscale for a given color.

Parameters

- **color** (*ColorTuple*) – The color to make a solid colors from
- **min_scale** (*number*) – The minimum value on this colorscale
- **max_scale** (*number*) – The maximum value on this colorscale

Returns A solid colorscale

Return type Tuple[Tuple[number, str]]

CHAPTER

THREE

GEOHEXVIZ - ERRORS MODULE

This module consists of the custom error types used by GeoHexViz's Plot Builder.

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**(*name: str*, *dstype:* geohexviz.errors.LayerType, *message: str = 'There was an error while reading the data file.'*)

exception geohexviz.errors.**DataReadError**(*name: str*, *dstype:* geohexviz.errors.LayerType, *message: str = "There was an error while reading the 'data' parameter passed."*)

exception geohexviz.errors.**DataTypeError**(*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 layer 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

FOUR

GEOHEXVIZ - TEMPLATES MODULE

This module consists of the templates that GeoHexViz's PlotBuilder uses for its layers and internal figure.

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

FIVE

FURTHER INFORMATION

Welcome to GeoHexViz!

Creating geospatial visualizations is often time-consuming and laborious. For example, an analyst must make a variety of design decisions, including which map projection to use, the colour scheme, the basemap, and how to organize the data in layers. One of many software applications may be used to construct the visualization, such as:

- **ArcGIS** which provides a wide range of capabilities, but requires a paid license and a solid foundation in geospatial information processing;
- **QGIS** which is free and open source, but like ArcGIS requires in-depth knowledge of geospatial information processing to be used effectively;
- **D3** which emphasizes web standards rather than a proprietary framework, but requires extensive knowledge of JavaScript; and
- **Plotly** which is a free and open source Python graphing library, but like D3 and other packages requires knowledge of a programming language.

Common across these applications is the requirement to have knowledge of geospatial concepts, and acquiring this knowledge has been identified as a significant challenge. In addition, the latter two options require programming. While many analysts have programming experience, not all do and in time-sensitive situations, as often encountered in a military setting, writing code to produce a visualization may not be feasible. With this in mind, GeoHexViz aims to reduce the time, in-depth knowledge, and programming required to produce publication-quality geospatial visualizations that use hexagonal binning. Implemented in Python, it seamlessly integrates several underlying packages &emdash; Pandas, GeoPandas, Uber H3, Shapely, and Plotly &emdash; and extends their functionality to achieve these goals. Although originally designed for use within the military operations research community, it may be used in other research communities.

5.1 Example Usage

GeoHexViz allows a user to generate hexagonally binned geospatial visualizations with two different methods. Method 1 concerns using the GeoHexSimple package's script to run a file containing plot structure. Method 2 concerns using Python code to interact with the functions within the package. Method 2 method has two categories:

a) Using functions from the GeoHexSimple package

b) Using functions from the GeoHexViz package

Please refer to the examples directory for additional examples that go into great depth (for both methods).

5.1.1 Method 1 Example Usage

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

```
{
  "hexbin_layer": {
    "data": "<sample csv file>",
    "hex_resolution": 4
  },
  "output": {
    "filepath": "<sample filepath>",
    "width": 600,
    "height": 400
  },
  "display": true
}
```

Running the JSON script will allow you to input a JSON file via command-line. The GeoHexSimple command-line script was created using argparse and is very robust. Running the help command provides the following:

```
>geohexsimple --help
usage: geohexsimple [options]

Input plot property files to make hexagonally binned plots.

optional arguments:
  -h, --help            show this help message and exit
  -p PATH, --path PATH  path to json file or directory containing json files (required␣
↪if no gui is used)
  -g, --gui             enable command-line gui (set to true if no path is provided)
  -nf, --nofeedback     turn off feedback while plotting
  -v, --verbose         whether to raise all errors or not
```

Running your plot properties file may look something like:

```
>geohexsimple --path <path to file>
exit
```

Or something like:

```
>geohexsimple

================GeoHexSimple===============
 A script for the simple creation of
 hexagonally binned geospatial visualizations.
===========================================
Main Menu
Please input the location of your parameterized
builder file (JSON, YAML) or a directory containing
builder files.
Options: file path, help, exit.
<path to file>
```

5.1.2 Method 2

As previously mentioned there are two ways to use the GeoHexViz library in Python code. Method 2a concerns using the functions that the GeoHexSimple script uses to create plots from pre-existing plot parameter files. Method 2b concerns using the functions from the GeoHexViz package to create plots.

Method 2a Example Usage

You can use the functions that the GeoHexSimple script uses to create a plot from a pre-existing plot parameter file. A simple example of this method is given below.

```
from geohexviz.utils.file import run_json

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

Method 2b Example Usage

You can use the functions and objects within GeoHexViz to create a plot from scratch. A simple example of this method is given below.

```
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],
    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 Installation

There are a few steps that a user must follow when installing GeoHexViz. First, the user must install GeoPandas. This is most easily done through the use of Anaconda, with this tool it can be installed like this:

```
conda install -c conda-forge geopandas
```

The version that GeoHexViz was developed with is version 0.8.1 (build py_0). Next, the user must download or clone GeoHexViz's GitHub repository. Finally, the user can navigate to the directory containing the `setup.py` file, and run:

```
python setup.py install
```

Or

```
pip install .
```

Note that to use the pdf cropping features, the user can do an editable install:

```
pip install -e .[pdf-crop]
```

The user may also install using pip and GitHub:

```
pip install git+https://github.com/tony-zeidan/geohexviz.git
```

Setting up a conda environment first helps. To make this process smoother the `environment.yml` file is included, which includes all dependencies. With this file, the first step (installation of GeoPandas) is done automatically. Using this file an environment can be set up like this:

```
conda env create -f environment.yml
```

This will create an Anaconda environment called `geohexviz` on your machine, simply activate the environment and run the `setup.py` file as shown above.

5.3 Further Documentation

The official documentation for GeoHexViz can be found at this page. The reference document published alongside this package can also be seen in the docs directory.

5.4 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.

5.5 Contributing

For major changes, please open an issue first to discuss what you would like to change. For more details please see this page.

5.6 Acknowledgements

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

5.7 Contact

For any questions, feedback, bug reports, feature requests, etc please first present your thoughts via GitHub issues. For further assistance please contact tony.azp25@gmail.com.

5.8 Copyright and License

For license see this page.

PYTHON MODULE INDEX

g

INDEX

G

H

L

M

N

O

P

R

S

U