PyGraphistry Documentation

Graphistry, Inc.

Apr 30, 2021

CONTENTS

1 graphistry package **3**
- 1.1 Submodules . . . 3
- 1.2 graphistry.plotter module . . . 3
- 1.3 graphistry.pygraphistry module . . . 16
- 1.4 graphistry.arrow_uploader module . . . 45
- 1.5 graphistry.ArrowFileUploader module . . . 46

2 doc **49**
- 2.1 versioneer module . . . 49

3 Indices and tables **51**

Index **53**

Quickstart: Read our tutorial

CHAPTER

ONE

GRAPHISTRY PACKAGE

1.1 Submodules

1.2 graphistry.plotter module

class graphistry.plotter.**Plotter**

Bases: object

Graph plotting class.

Created using `Graphistry.bind()`.

Chained calls successively add data and visual encodings, and end with a plot call.

To streamline reuse and replayable notebooks, Plotter manipulations are immutable. Each chained call returns a new instance that derives from the previous one. The old plotter or the new one can then be used to create different graphs.

When using memoization, for .register(api=3) sessions with .plot(memoize=True), Pandas/cudf arrow coercions are memoized, and file uploads are skipped on same-hash dataframes.

The class supports convenience methods for mixing calls across Pandas, NetworkX, and IGraph.

addStyle(*fg=None*, *bg=None*, *page=None*, *logo=None*)

Set general visual styles

See .bind() and .settings(url_params={}) for additional styling options, and style() for another way to set the same attributes.

To facilitate reuse and replayable notebooks, the addStyle() call is chainable. Invocation does not effect the old style: it instead returns a new Plotter instance with the new styles added to the existing ones. Both the old and new styles can then be used for different graphs.

addStyle() will extend the existing style settings, while style() will replace any in the same group

Parameters

- **fg** (*dict*) – Dictionary {'blendMode': str} of any valid CSS blend mode
- **bg** (*dict*) – Nested dictionary of page background properties. {'color': str, 'gradient': {'kind': str, 'position': str, 'stops': list }, 'image': { 'url': str, 'width': int, 'height': int, 'blendMode': str }
- **logo** (*dict*) – Nested dictionary of logo properties. { 'url': str, 'autoInvert': bool, 'position': str, 'dimensions': { 'maxWidth': int, 'maxHeight': int }, 'crop': { 'top': int, 'left': int, 'bottom': int, 'right': int }, 'padding': { 'top': int, 'left': int, 'bottom': int, 'right': int}, 'style': str}

- **page** (*dict*) – Dictionary of page metadata settings. { 'favicon': str, 'title': str }

Returns Plotter

Return type *Plotter*

Example: Chained merge - results in color, blendMode, and url being set

```
g2 =  g.addStyle(bg={'color': 'black'}, fg={'blendMode': 'screen'})
g3 = g2.addStyle(bg={'image': {'url': 'http://site.com/watermark.png'}})
```

Example: Overwrite - results in blendMode multiply

```
g2 =  g.addStyle(fg={'blendMode': 'screen'})
g3 = g2.addStyle(fg={'blendMode': 'multiply'})
```

Example: Gradient background

```
g.addStyle(bg={'gradient': {'kind': 'linear', 'position': 45, 'stops': [[
↪'rgb(0,0,0)', '0%'], ['rgb(255,255,255)', '100%']]}})
```

Example: Page settings

```
g.addStyle(page={'title': 'Site - {{ name }}', 'favicon': 'http://site.
↪com/logo.ico'})
```

bind(*source=None*, *destination=None*, *node=None*, *edge_title=None*, *edge_label=None*, *edge_color=None*, *edge_weight=None*, *edge_size=None*, *edge_opacity=None*, *edge_icon=None*, *edge_source_color=None*, *edge_destination_color=None*, *point_title=None*, *point_label=None*, *point_color=None*, *point_weight=None*, *point_size=None*, *point_opacity=None*, *point_icon=None*, *point_x=None*, *point_y=None*)

Relate data attributes to graph structure and visual representation.

To facilitate reuse and replayable notebooks, the binding call is chainable. Invocation does not effect the old binding: it instead returns a new Plotter instance with the new bindings added to the existing ones. Both the old and new bindings can then be used for different graphs.

Parameters

- **source** (*str*) – Attribute containing an edge's source ID
- **destination** (*str*) – Attribute containing an edge's destination ID
- **node** (*str*) – Attribute containing a node's ID
- **edge_title** (*str*) – Attribute overriding edge's minimized label text. By default, the edge source and destination is used.
- **edge_label** (*str*) – Attribute overriding edge's expanded label text. By default, scrollable list of attribute/value mappings.
- **edge_color** (*str*) – Attribute overriding edge's color. rgba (int64) or int32 palette index, see palette definitions <https://graphistry.github.io/docs/legacy/api/0.9.2/api.html#extendedpalette>`_ for values. Based on Color Brewer.
- **edge_source_color** (*str*) – Attribute overriding edge's source color if no edge_color, as an rgba int64 value.
- **edge_destination_color** (*str*) – Attribute overriding edge's destination color if no edge_color, as an rgba int64 value.

- **edge_weight** (*str*) – Attribute overriding edge weight. Default is 1. Advanced layout controls will relayout edges based on this value.
- **point_title** (*str*) – Attribute overriding node's minimized label text. By default, the node ID is used.
- **point_label** (*str*) – Attribute overriding node's expanded label text. By default, scrollable list of attribute/value mappings.
- **point_color** (*str*) – Attribute overriding node's color.rgba (int64) or int32 palette index, see palette definitions <https://graphistry.github.io/docs/legacy/api/0.9.2/api.html#extendedpalette>`_ for values. Based on Color Brewer.
- **point_size** (*str*) – Attribute overriding node's size. By default, uses the node degree. The visualization will normalize point sizes and adjust dynamically using semantic zoom.
- **point_x** (*str*) – Attribute overriding node's initial x position. Combine with ".settings(url_params={'play': 0}))" to create a custom layout
- **point_y** (*str*) – Attribute overriding node's initial y position. Combine with ".settings(url_params={'play': 0}))" to create a custom layout

Returns Plotter

Return type *Plotter*

Example: Minimal

```
import graphistry
g = graphistry.bind()
g = g.bind(source='src', destination='dst')
```

Example: Node colors

```
import graphistry
g = graphistry.bind()
g = g.bind(source='src', destination='dst',
           node='id', point_color='color')
```

Example: Chaining

```
import graphistry
g = graphistry.bind(source='src', destination='dst', node='id')

g1 = g.bind(point_color='color1', point_size='size1')

g.bind(point_color='color1b')

g2a = g1.bind(point_color='color2a')
g2b = g1.bind(point_color='color2b', point_size='size2b')

g3a = g2a.bind(point_size='size3a')
g3b = g2b.bind(point_size='size3b')
```

In the above Chaining example, all bindings use src/dst/id. Colors and sizes bind to:

```
g: default/default
g1: color1/size1
g2a: color2a/size1
g2b: color2b/size2b
```

(continues on next page)

(continued from previous page)

```
g3a: color2a/size3a
g3b: color2b/size3b
```

bolt(*driver*)

cypher(*query*, *params={}*)

description(*description*)
Upload description

Parameters **description** (*str*) – Upload description

edges(*edges*, *source=None*, *destination=None*)
Specify edge list data and associated edge attribute values.

Parameters **edges** – Edges and their attributes.

Returns Plotter

Return type *Plotter*

Example

```
import graphistry
df = pandas.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
graphistry
    .bind(source='src', destination='dst')
    .edges(df)
    .plot()
```

Example

```
import graphistry
df = pandas.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
graphistry
    .edges(df, 'src', 'dst')
    .plot()
```

encode_edge_badge(*column*, *position='TopRight'*, *categorical_mapping=None*, *continuous_binning=None*, *default_mapping=None*, *comparator=None*, *color=None*, *bg=None*, *fg=None*, *for_current=False*, *for_default=True*, *as_text=None*, *blend_mode=None*, *style=None*, *border=None*, *shape=None*)

encode_edge_color(*column*, *palette=None*, *as_categorical=None*, *as_continuous=None*, *categorical_mapping=None*, *default_mapping=None*, *for_default=True*, *for_current=False*)
Set edge color with more control than bind()

Parameters

- **column** (*str*) – Data column name
- **palette** (*Optional[list]*) – Optional list of color-like strings. Ex: [“black, “#FF0”, “rgb(255,255,255)”]. Used as a gradient for continuous and round-robin for categorical.
- **as_categorical** (*Optional[bool]*) – Interpret column values as categorical. Ex: Uses palette via round-robin when more values than palette entries.

- **as_continuous** (*Optional[bool]*) – Interpret column values as continuous. Ex: Uses palette for an interpolation gradient when more values than palette entries.
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to color-like strings. Ex: {"car": "red", "truck": #000"}
- **default_mapping** (*Optional[str]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping="gray".
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.

Returns Plotter

Return type *Plotter*

Example: See encode_point_color

encode_edge_icon(*column*, *categorical_mapping=None*, *continuous_binning=None*, *default_mapping=None*, *comparator=None*, *for_default=True*, *for_current=False*, *as_text=False*, *blend_mode=None*, *style=None*, *border=None*, *shape=None*)

Set edge icon with more control than bind() Values from Font Awesome 4 such as "laptop": https://fontawesome.com/v4.7.0/icons/ , image URLs (http://...), and data URIs (data:...). When as_text=True is enabled, values are instead interpreted as raw strings.

Parameters

- **column** (*str*) – Data column name
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to icon name strings. Ex: {"toyota": 'car', "ford": 'truck'}
- **default_mapping** (*Optional[Union[int,float]]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=50.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.
- **as_text** (*Optional[bool]*) – Values should instead be treated as raw strings, instead of icons and images. (Default False.)

Returns Plotter

Return type *Plotter*

Example: Set a string column of icons for the edge icons, same as bind(edge_icon='my_column')

```
g2a = g.encode_edge_icon('my_icons_column')
```

Example: Map specific values to specific icons, including with a default

```
g2a = g.encode_edge_icon('brands', categorical_mapping={'toyota': 'car',
↪'ford': 'truck'})
g2b = g.encode_edge_icon('brands', categorical_mapping={'toyota': 'car',
↪'ford': 'truck'}, default_mapping='question')
```

Example: Map countries to abbreviations

```
g2a = g.encode_edge_icon('country_abbrev', as_text=True)
g2b = g.encode_edge_icon('country', as_text=True, categorical_mapping={
↪'England': 'UK', 'America': 'US'}, default_mapping='')
```

Example: Border

```
g2b = g.encode_edge_icon('country', border={'width': 3, color: 'black',
↪'stroke': 'dashed'}, 'categorical_mapping={'England': 'UK', 'America':
↪'US'})
```

encode_point_badge(*column*, *position='TopRight'*, *categorical_mapping=None*, *continuous_binning=None*, *default_mapping=None*, *comparator=None*, *color=None*, *bg=None*, *fg=None*, *for_current=False*, *for_default=True*, *as_text=None*, *blend_mode=None*, *style=None*, *border=None*, *shape=None*)

encode_point_color(*column*, *palette=None*, *as_categorical=None*, *as_continuous=None*, *categorical_mapping=None*, *default_mapping=None*, *for_default=True*, *for_current=False*)

Set point color with more control than bind()

Parameters

- **column** (*str*) – Data column name
- **palette** (*Optional[list]*) – Optional list of color-like strings. Ex: [“black, “#FF0”, “rgb(255,255,255)”]. Used as a gradient for continuous and round-robin for categorical.
- **as_categorical** (*Optional[bool]*) – Interpret column values as categorical. Ex: Uses palette via round-robin when more values than palette entries.
- **as_continuous** (*Optional[bool]*) – Interpret column values as continuous. Ex: Uses palette for an interpolation gradient when more values than palette entries.
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to color-like strings. Ex: {“car”: “red”, “truck”: #000”}
- **default_mapping** (*Optional[str]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=”gray”.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.

Returns Plotter

Return type *Plotter*

Example: Set a palette-valued column for the color, same as bind(point_color=’my_column’)

```
g2a = g.encode_point_color('my_int32_palette_column')
g2b = g.encode_point_color('my_int64_rgb_column')
```

Example: Set a cold-to-hot gradient of along the spectrum blue, yellow, red

```
g2 = g.encode_point_color('my_numeric_col', palette=["blue", "yellow",
↪"red"], as_continuous=True)
```

Example: Round-robin sample from 5 colors in hex format

```
g2 = g.encode_point_color('my_distinctly_valued_col', palette=["#000", "
↪#00F", "#0F0", "#0FF", "#FFF"], as_categorical=True)
```

Example: Map specific values to specific colors, including with a default

```
g2a = g.encode_point_color('brands', categorical_mapping={'toyota': 'red',
↪ 'ford': 'blue'})
g2a = g.encode_point_color('brands', categorical_mapping={'toyota': 'red',
↪ 'ford': 'blue'}, default_mapping='gray')
```

encode_point_icon(*column*, *categorical_mapping=None*, *continuous_binning=None*, *default_mapping=None*, *comparator=None*, *for_default=True*, *for_current=False*, *as_text=False*, *blend_mode=None*, *style=None*, *border=None*, *shape=None*)

Set node icon with more control than bind(). Values from Font Awesome 4 such as "laptop": https://fontawesome.com/v4.7.0/icons/ , image URLs (http://...), and data URIs (data:...). When as_text=True is enabled, values are instead interpreted as raw strings.

Parameters

- **column** (*str*) – Data column name
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to icon name strings. Ex: {"toyota": 'car', "ford": 'truck'}
- **default_mapping** (*Optional[Union[int,float]]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=50.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.
- **as_text** (*Optional[bool]*) – Values should instead be treated as raw strings, instead of icons and images. (Default False.)
- **blend_mode** (*Optional[str]*) – CSS blend mode
- **style** (*Optional[dict]*) – CSS filter properties - opacity, saturation, luminosity, grayscale, and more
- **border** (*Optional[dict]*) – Border properties - 'width', 'color', and 'storke'

Returns Plotter

Return type *Plotter*

Example: Set a string column of icons for the point icons, same as bind(point_icon='my_column')

```
g2a = g.encode_point_icon('my_icons_column')
```

Example: Map specific values to specific icons, including with a default

```
g2a = g.encode_point_icon('brands', categorical_mapping={'toyota': 'car',
↪'ford': 'truck'})
g2b = g.encode_point_icon('brands', categorical_mapping={'toyota': 'car',
↪'ford': 'truck'}, default_mapping='question')
```

Example: Map countries to abbreviations

```
g2b = g.encode_point_icon('country_abbrev', as_text=True)
g2b = g.encode_point_icon('country', as_text=True, categorical_mapping={
↪'England': 'UK', 'America': 'US'}, default_mapping='')
```

Example: Border

```
g2b = g.encode_point_icon('country', border={'width': 3, color: 'black',
↪'stroke': 'dashed'}, 'categorical_mapping={'England': 'UK', 'America':
↪'US'})
```

encode_point_size(*column*, *categorical_mapping=None*, *default_mapping=None*, *for_default=True*, *for_current=False*)

Set point size with more control than bind()

Parameters

- **column** (*str*) – Data column name
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to numbers. Ex: {"car": 100, "truck": 200}
- **default_mapping** (*Optional[Union[int,float]]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=50.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.

Returns Plotter

Return type *Plotter*

Example: Set a numerically-valued column for the size, same as bind(point_size='my_column')

```
g2a = g.encode_point_size('my_numeric_column')
```

Example: Map specific values to specific colors, including with a default

```
g2a = g.encode_point_size('brands', categorical_mapping={'toyota': 100,
↪'ford': 200})
g2b = g.encode_point_size('brands', categorical_mapping={'toyota': 100,
↪'ford': 200}, default_mapping=50)
```

graph(*ig*)

Specify the node and edge data.

Parameters **ig** (*Any*) – NetworkX graph or an IGraph graph with node and edge attributes.

Returns Plotter

Return type *Plotter*

gsql(*query*, *bindings={}*, *dry_run=False*)

Run Tigergraph query in interpreted mode and return transformed Plottable

param query Code to run

type query str

param bindings Mapping defining names of returned 'edges' and/or 'nodes', defaults to @@nodeList and @@edgeList

type bindings Optional[dict]

param dry_run Return target URL without running

type dry_run bool

returns Plotter

rtype Plotter

Example: Minimal

```
import graphistry
tg = graphistry.tigergraph()
tg.gsql("""
INTERPRET QUERY () FOR GRAPH Storage {

    OrAccum<BOOL> @@stop;
    ListAccum<EDGE> @@edgeList;
    SetAccum<vertex> @@set;

    @@set += to_vertex("61921", "Pool");

    Start = @@set;

    while Start.size() > 0 and @@stop == false do

    Start = select t from Start:s-(:e)-:t
    where e.goUpper == TRUE
    accum @@edgeList += e
    having t.type != "Service";
    end;

    print @@edgeList;
}
""").plot()
```

Example: Full

```
import graphistry
tg = graphistry.tigergraph()
tg.gsql("""
INTERPRET QUERY () FOR GRAPH Storage {

    OrAccum<BOOL> @@stop;
    ListAccum<EDGE> @@edgeList;
    SetAccum<vertex> @@set;
```

(continues on next page)

(continued from previous page)

```
    @@set += to_vertex("61921", "Pool");

    Start = @@set;

    while Start.size() > 0 and @@stop == false do

    Start = select t from Start:s-(:e)-:t
    where e.goUpper == TRUE
    accum @@edgeList += e
    having t.type != "Service";
    end;

    print @@my_edge_list;
}
""", {'edges': 'my_edge_list'}).plot()
```

gsql_endpoint(*method_name*, *args={}*, *bindings={}*, *db=None*, *dry_run=False*)

Invoke Tigergraph stored procedure at a user-definend endpoint and return transformed Plottable

Parameters

- **method_name** (*str*) – Stored procedure name
- **args** (*Optional[dict]*) – Named endpoint arguments
- **bindings** (*Optional[dict]*) – Mapping defining names of returned 'edges' and/or 'nodes', defaults to @@nodeList and @@edgeList
- **db** (*Optional[str]*) – Name of the database, defaults to value set in .tigergraph(...)
- **dry_run** (*bool*) – Return target URL without running

Returns Plotter

Return type *Plotter*

Example: Minimal

```
import graphistry
tg = graphistry.tigergraph(db='my_db')
tg.gsql_endpoint('neighbors').plot()
```

Example: Full

```
import graphistry
tg = graphistry.tigergraph()
tg.gsql_endpoint('neighbors', {'k': 2}, {'edges': 'my_edge_list'}, 'my_db
↪').plot()
```

Example: Read data

```
import graphistry
tg = graphistry.tigergraph()
out = tg.gsql_endpoint('neighbors')
(nodes_df, edges_df) = (out._nodes, out._edges)
```

igraph2pandas(*ig*)

Under current bindings, transform an IGraph into a pandas edges dataframe and a nodes dataframe.

Example

```
import graphistry
g = graphistry.bind()

es = pandas.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
g = g.bind(source='src', destination='dst').edges(es)

ig = g.pandas2igraph(es)
ig.vs['community'] = ig.community_infomap().membership

(es2, vs2) = g.igraph2pandas(ig)
g.nodes(vs2).bind(point_color='community').plot()
```

name (*name*)

Upload name

Parameters **name** (*str*) – Upload name

networkx2pandas (*g*)

networkx_checkoverlap (*g*)

nodes (*nodes*, *node=None*)

Specify the set of nodes and associated data.

Must include any nodes referenced in the edge list.

Parameters **nodes** – Nodes and their attributes.

Returns Plotter

Return type *Plotter*

Example

```
import graphistry

es = pandas.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
g = graphistry
    .bind(source='src', destination='dst')
    .edges(es)

vs = pandas.DataFrame({'v': [0,1,2], 'lbl': ['a', 'b', 'c']})
g = g.bind(node='v').nodes(vs)

g.plot()
```

Example

```
import graphistry

es = pandas.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
g = graphistry.edges(es, 'src', 'dst')

vs = pandas.DataFrame({'v': [0,1,2], 'lbl': ['a', 'b', 'c']})
g = g.nodes(vs, 'v)

g.plot()
```

nodexl (*xls_or_url*, *source='default'*, *engine=None*, *verbose=False*)

pandas2igraph(*edges*, *directed=True*)
Convert a pandas edge dataframe to an IGraph graph.

Uses current bindings. Defaults to treating edges as directed.

Example

```
import graphistry
g = graphistry.bind()

es = pandas.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
g = g.bind(source='src', destination='dst')

ig = g.pandas2igraph(es)
ig.vs['community'] = ig.community_infomap().membership
g.bind(point_color='community').plot(ig)
```

plot(*graph=None*, *nodes=None*, *name=None*, *description=None*, *render=None*, *skip_upload=False*, *as_files=False*, *memoize=True*)
Upload data to the Graphistry server and show as an iframe of it.

Uses the currently bound schema structure and visual encodings. Optional parameters override the current bindings.

When used in a notebook environment, will also show an iframe of the visualization.

Parameters

- **graph** (*Any*) – Edge table (pandas, arrow, cudf) or graph (NetworkX, IGraph).
- **nodes** (*Any*) – Nodes table (pandas, arrow, cudf)
- **name** (*str*) – Upload name.
- **description** (*str*) – Upload description.
- **render** (*bool*) – Whether to render the visualization using the native notebook environment (default True), or return the visualization URL
- **skip_upload** (*bool*) – Return node/edge/bindings that would have been uploaded. By default, upload happens.
- **as_files** (*bool*) – Upload distinct node/edge files under the managed Files PI. Default off, will switch to default-on when stable.
- **memoize** (*bool*) – Tries to memoize pandas/cudf->arrow conversion, including skipping upload. Default on.

Example: Simple

```
import graphistry
es = pandas.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
graphistry
    .bind(source='src', destination='dst')
    .edges(es)
    .plot()
```

Example: Shorthand

```
import graphistry
es = pandas.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
graphistry
```

(continues on next page)

(continued from previous page)

```
    .bind(source='src', destination='dst')
    .plot(es)
```

settings (*height=None*, *url_params={}*, *render=None*)
Specify iframe height and add URL parameter dictionary.

The library takes care of URI component encoding for the dictionary.

Parameters

- **height** (*int*) – Height in pixels.
- **url_params** (*dict*) – Dictionary of querystring parameters to append to the URL.
- **render** (*bool*) – Whether to render the visualization using the native notebook environment (default True), or return the visualization URL

style (*fg=None*, *bg=None*, *page=None*, *logo=None*)
Set general visual styles

See .bind() and .settings(url_params={}) for additional styling options, and addStyle() for another way to set the same attributes.

To facilitate reuse and replayable notebooks, the style() call is chainable. Invocation does not effect the old style: it instead returns a new Plotter instance with the new styles added to the existing ones. Both the old and new styles can then be used for different graphs.

style() will fully replace any defined parameter in the existing style settings, while addStyle() will merge over previous values

Parameters

- **fg** (*dict*) – Dictionary {'blendMode': str} of any valid CSS blend mode
- **bg** (*dict*) – Nested dictionary of page background properties. {'color': str, 'gradient': {'kind': str, 'position': str, 'stops': list }, 'image': { 'url': str, 'width': int, 'height': int, 'blendMode': str }
- **logo** (*dict*) – Nested dictionary of logo properties. { 'url': str, 'autoInvert': bool, 'position': str, 'dimensions': { 'maxWidth': int, 'maxHeight': int }, 'crop': { 'top': int, 'left': int, 'bottom': int, 'right': int }, 'padding': { 'top': int, 'left': int, 'bottom': int, 'right': int}, 'style': str}
- **page** (*dict*) – Dictionary of page metadata settings. { 'favicon': str, 'title': str }

Returns Plotter

Return type *Plotter*

Example: Chained merge - results in url and blendMode being set, while color is dropped

:: g2 = g.style(bg={'color': 'black'}, fg={'blendMode': 'screen'}) g3 = g2.style(bg={'image': {'url': 'http://site.com/watermark.png'}})

Example: Gradient background

:: g.style(bg={'gradient': {'kind': 'linear', 'position': 45, 'stops': [['rgb(0,0,0)', '0%'], ['rgb(255,255,255)', '100%']]}})

Example: Page settings

:: g.style(page={'title': 'Site - {{ name }}', 'favicon': 'http://site.com/logo.ico'})

tigergraph(*protocol='http'*, *server='localhost'*, *web_port=14240*, *api_port=9000*, *db=None*, *user='tigergraph'*, *pwd='tigergraph'*, *verbose=False*)

Register Tigergraph connection setting defaults

Parameters

- **protocol** (*Optional[str]*) – Protocol used to contact the database.
- **server** (*Optional[str]*) – Domain of the database
- **web_port** (*Optional[int]*) –
- **api_port** (*Optional[int]*) –
- **db** (*Optional[str]*) – Name of the database
- **user** (*Optional[str]*) –
- **pwd** (*Optional[str]*) –
- **verbose** (*Optional[bool]*) – Whether to print operations

Returns Plotter

Return type *Plotter*

Example: Standard

```
import graphistry
tg = graphistry.tigergraph(protocol='https', server='acme.com', db='my_db
↪', user='alice', pwd='tigergraph2')
```

class graphistry.plotter.**WeakValueWrapper**(*v*)

Bases: object

graphistry.plotter.**cache_coercion**(*k*, *v*)

Holds references to last 100 used coercions Use with weak key/value dictionaries for actual lookups

graphistry.plotter.**cache_coercion_helper**(*k*)

1.3 graphistry.pygraphistry module

class graphistry.pygraphistry.**NumpyJSONEncoder**(***, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: json.encoder.JSONEncoder

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a TypeError).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
```

(continues on next page)

(continued from previous page)

```
            return list(iterable)
        # Let the base class default method raise the TypeError
        return JSONEncoder.default(self, o)
```

class graphistry.pygraphistry.**PyGraphistry**

Bases: object

static addStyle(*bg=None*, *fg=None*, *logo=None*, *page=None*)

Creates a base plotter with some style settings.

For parameters, see `plotter.addStyle`.

Returns Plotter

Return type *Plotter*

Example

```
import graphistry
graphistry.addStyle(bg={'color': 'black'})
```

static api_key(*value=None*)

Set or get the API key. Also set via environment variable GRAPHISTRY_API_KEY.

static api_token(*value=None*)

Set or get the API token. Also set via environment variable GRAPHISTRY_API_TOKEN.

static api_token_refresh_ms(*value=None*)

Set or get the API token refresh interval in milliseconds. None and 0 interpreted as no refreshing.

static api_version(*value=None*)

Set or get the API version: 1 or 2 for 1.0 (deprecated), 3 for 2.0 Also set via environment variable GRAPHISTRY_API_VERSION.

static authenticate()

Authenticate via already provided configuration (api=1,2). This is called once automatically per session when uploading and rendering a visualization. In api=3, if token_refresh_ms > 0 (defaults to 10min), this starts an automatic refresh loop. In that case, note that a manual .login() is still required every 24hr by default.

static bind(*node=None*, *source=None*, *destination=None*, *edge_title=None*, *edge_label=None*, *edge_color=None*, *edge_weight=None*, *edge_icon=None*, *edge_size=None*, *edge_opacity=None*, *edge_source_color=None*, *edge_destination_color=None*, *point_title=None*, *point_label=None*, *point_color=None*, *point_weight=None*, *point_icon=None*, *point_size=None*, *point_opacity=None*, *point_x=None*, *point_y=None*)

Create a base plotter.

Typically called at start of a program. For parameters, see `plotter.bind()` .

Returns Plotter

Return type *Plotter*

Example

```
import graphistry
g = graphistry.bind()
```

static bolt(*driver=None*)

Parameters **driver** – Neo4j Driver or arguments for GraphDatabase.driver(**{...}**)

Returns Plotter w/neo4j

Call this to create a Plotter with an overridden neo4j driver.

Example

```
import graphistry
g = graphistry.bolt({ server: 'bolt://...', auth: ('<username>', '
↪<password>') })
```

```
import neo4j
import graphistry

driver = neo4j.GraphDatabase.driver(...)

g = graphistry.bolt(driver)
```

static certificate_validation(*value=None*)

Enable/Disable SSL certificate validation (True, False). Also set via environment variable GRAPHISTRY_CERTIFICATE_VALIDATION.

static client_protocol_hostname(*value=None*)

Get/set the client protocol+hostname for when display urls (distinct from uploading). Also set via environment variable GRAPHISTRY_CLIENT_PROTOCOL_HOSTNAME. Defaults to hostname and no protocol (reusing environment protocol)

static cypher(*query*, *params={}*)

Parameters

- **query** – a cypher query
- **params** – cypher query arguments

Returns Plotter with data from a cypher query. This call binds *source*, *destination*, and *node*.

Call this to immediately execute a cypher query and store the graph in the resulting Plotter.

```
import graphistry
g = graphistry.bolt({ query='MATCH (a)-[r:PAYMENT]->(b) WHERE r.USD >␣
↪7000 AND r.USD < 10000 RETURN r ORDER BY r.USD DESC', params={
↪"AccountId": 10 })
```

static description(*description*)

Upload description

Parameters **description** (*str*) – Upload description

static edges(*edges*, *source=None*, *destination=None*)

Specify edge list data and associated edge attribute values.

Parameters **edges** – Edges and their attributes (Pandas dataframe, NetworkX graph, or IGraph graph)

Returns Plotter

Return type *Plotter*

Example

```
import graphistry
df = pd.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
graphistry
    .bind(source='src', destination='dst')
    .edges(df)
    .plot()
```

Example

```
import graphistry
df = pd.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
graphistry
    .edges(df, 'src', 'dst')
    .plot()
```

static encode_edge_badge(*column*, *position='TopRight'*, *categorical_mapping=None*, *continuous_binning=None*, *default_mapping=None*, *comparator=None*, *color=None*, *bg=None*, *fg=None*, *for_current=False*, *for_default=True*, *as_text=None*, *blend_mode=None*, *style=None*, *border=None*, *shape=None*)

static encode_edge_color(*column*, *palette=None*, *as_categorical=None*, *as_continuous=None*, *categorical_mapping=None*, *default_mapping=None*, *for_default=True*, *for_current=False*)

Set edge color with more control than bind()

Parameters

- **column** (*str*) – Data column name
- **palette** (*Optional[list]*) – Optional list of color-like strings. Ex: [“black, “#FF0”, “rgb(255,255,255)”]. Used as a gradient for continuous and round-robin for categorical.
- **as_categorical** (*Optional[bool]*) – Interpret column values as categorical. Ex: Uses palette via round-robin when more values than palette entries.
- **as_continuous** (*Optional[bool]*) – Interpret column values as continuous. Ex: Uses palette for an interpolation gradient when more values than palette entries.
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to color-like strings. Ex: {“car”: “red”, “truck”: #000”}
- **default_mapping** (*Optional[str]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=”gray”.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.

Returns Plotter

Return type *Plotter*

Example: See encode_point_color

static encode_edge_icon(*column*, *categorical_mapping=None*, *continuous_binning=None*, *default_mapping=None*, *comparator=None*, *for_default=True*, *for_current=False*, *as_text=False*, *blend_mode=None*, *style=None*, *border=None*, *shape=None*)

Set edge icon with more control than bind(). Values from Font Awesome 4 such as "laptop": https://fontawesome.com/v4.7.0/icons/

Parameters

- **column** (*str*) – Data column name
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to icon name strings. Ex: {"toyota": 'car', "ford": 'truck'}
- **default_mapping** (*Optional[Union[int,float]]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=50.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.
- **as_text** (*Optional[bool]*) – Values should instead be treated as raw strings, instead of icons and images. (Default False.)
- **blend_mode** (*Optional[str]*) – CSS blend mode
- **style** (*Optional[dict]*) – CSS filter properties - opacity, saturation, luminosity, grayscale, and more
- **border** (*Optional[dict]*) – Border properties - 'width', 'color', and 'storke'

Returns Plotter

Return type *Plotter*

Example: Set a string column of icons for the edge icons, same as bind(edge_icon='my_column')

```
g2a = g.encode_edge_icon('my_icons_column')
```

Example: Map specific values to specific icons, including with a default

```
g2a = g.encode_edge_icon('brands', categorical_mapping={'toyota': 'car',
↪'ford': 'truck'})
g2b = g.encode_edge_icon('brands', categorical_mapping={'toyota': 'car',
↪'ford': 'truck'}, default_mapping='question')
```

Example: Map countries to abbreviations

```
g2a = g.encode_edge_icon('country_abbrev', as_text=True)
g2b = g.encode_edge_icon('country', categorical_mapping={'England': 'UK',
↪'America': 'US'}, default_mapping='')
```

Example: Border

```
g2b = g.encode_edge_icon('country', border={'width': 3, color: 'black',
↪'stroke': 'dashed'}, 'categorical_mapping={'England': 'UK', 'America':
↪'US'})
```

static encode_point_badge(*column*, *position='TopRight'*, *categorical_mapping=None*, *continuous_binning=None*, *default_mapping=None*, *comparator=None*, *color=None*, *bg=None*, *fg=None*, *for_current=False*, *for_default=True*, *as_text=None*, *blend_mode=None*, *style=None*, *border=None*, *shape=None*)

static encode_point_color(*column*, *palette=None*, *as_categorical=None*, *as_continuous=None*, *categorical_mapping=None*, *default_mapping=None*, *for_default=True*, *for_current=False*)

Set point color with more control than bind()

Parameters

- **column** (*str*) – Data column name
- **palette** (*Optional[list]*) – Optional list of color-like strings. Ex: [“black, “#FF0”, “rgb(255,255,255)”]. Used as a gradient for continuous and round-robin for categorical.
- **as_categorical** (*Optional[bool]*) – Interpret column values as categorical. Ex: Uses palette via round-robin when more values than palette entries.
- **as_continuous** (*Optional[bool]*) – Interpret column values as continuous. Ex: Uses palette for an interpolation gradient when more values than palette entries.
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to color-like strings. Ex: {“car”: “red”, “truck”: #000”}
- **default_mapping** (*Optional[str]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=”gray”.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.

Returns Plotter

Return type *Plotter*

Example: Set a palette-valued column for the color, same as bind(point_color='my_column')

```
g2a = g.encode_point_color('my_int32_palette_column')
g2b = g.encode_point_color('my_int64_rgb_column')
```

Example: Set a cold-to-hot gradient of along the spectrum blue, yellow, red

```
g2 = g.encode_point_color('my_numeric_col', palette=["blue", "yellow",
↪"red"], as_continuous=True)
```

Example: Round-robin sample from 5 colors in hex format

```
g2 = g.encode_point_color('my_distinctly_valued_col', palette=["#000", "
↪#00F", "#0F0", "#0FF", "#FFF"], as_categorical=True)
```

Example: Map specific values to specific colors, including with a default

```
g2a = g.encode_point_color('brands', categorical_mapping={'toyota': 'red',
↪ 'ford': 'blue'})
g2a = g.encode_point_color('brands', categorical_mapping={'toyota': 'red',
↪ 'ford': 'blue'}, default_mapping='gray')
```

static encode_point_icon(*column*, *categorical_mapping=None*, *continuous_binning=None*, *default_mapping=None*, *comparator=None*, *for_default=True*, *for_current=False*, *as_text=False*, *blend_mode=None*, *style=None*, *border=None*, *shape=None*)

Set node icon with more control than bind(). Values from Font Awesome 4 such as "laptop": https://fontawesome.com/v4.7.0/icons/

Parameters

- **column** (*str*) – Data column name
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to icon name strings. Ex: {"toyota": 'car', "ford": 'truck'}
- **default_mapping** (*Optional[Union[int,float]]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=50.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.
- **as_text** (*Optional[bool]*) – Values should instead be treated as raw strings, instead of icons and images. (Default False.)
- **blend_mode** (*Optional[str]*) – CSS blend mode
- **style** (*Optional[dict]*) – CSS filter properties - opacity, saturation, luminosity, grayscale, and more
- **border** (*Optional[dict]*) – Border properties - 'width', 'color', and 'storke'

Returns Plotter

Return type *Plotter*

Example: Set a string column of icons for the point icons, same as bind(point_icon='my_column')

```
g2a = g.encode_point_icon('my_icons_column')
```

Example: Map specific values to specific icons, including with a default

```
g2a = g.encode_point_icon('brands', categorical_mapping={'toyota': 'car',
↪'ford': 'truck'})
g2b = g.encode_point_icon('brands', categorical_mapping={'toyota': 'car',
↪'ford': 'truck'}, default_mapping='question')
```

Example: Map countries to abbreviations

```
g2b = g.encode_point_icon('country_abbrev', as_text=True)
g2b = g.encode_point_icon('country', as_text=True, categorical_mapping={
↪'England': 'UK', 'America': 'US'}, default_mapping='')
```

Example: Border

```
g2b = g.encode_point_icon('country', border={'width': 3, color: 'black',
↪'stroke': 'dashed'}, 'categorical_mapping={'England': 'UK', 'America':
↪'US'})
```

static encode_point_size(*column*, *categorical_mapping=None*, *default_mapping=None*, *for_default=True*, *for_current=False*)

Set point size with more control than bind()

Parameters

- **column** (*str*) – Data column name
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to numbers. Ex: {"car": 100, "truck": 200}
- **default_mapping** (*Optional[Union[int,float]]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=50.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.

Returns Plotter

Return type *Plotter*

Example: Set a numerically-valued column for the size, same as bind(point_size='my_column')

```
g2a = g.encode_point_size('my_numeric_column')
```

Example: Map specific values to specific colors, including with a default

```
g2a = g.encode_point_size('brands', categorical_mapping={'toyota': 100,
↪'ford': 200})
g2b = g.encode_point_size('brands', categorical_mapping={'toyota': 100,
↪'ford': 200}, default_mapping=50)
```

static graph(*ig*)

static gsql(*query*, *bindings=None*, *dry_run=False*)

Run Tigergraph query in interpreted mode and return transformed Plottable

param query Code to run

type query str

param bindings Mapping defining names of returned 'edges' and/or 'nodes', defaults to @@nodeList and @@edgeList

type bindings Optional[dict]

param dry_run Return target URL without running

type dry_run bool

returns Plotter

rtype Plotter

Example: Minimal

```
import graphistry
tg = graphistry.tigergraph()
tg.gsql("""
INTERPRET QUERY () FOR GRAPH Storage {

    OrAccum<BOOL> @@stop;
    ListAccum<EDGE> @@edgeList;
    SetAccum<vertex> @@set;
```

(continues on next page)

(continued from previous page)

```
    @@set += to_vertex("61921", "Pool");

    Start = @@set;

    while Start.size() > 0 and @@stop == false do

    Start = select t from Start:s-(:e)-:t
    where e.goUpper == TRUE
    accum @@edgeList += e
    having t.type != "Service";
    end;

    print @@edgeList;
}
""").plot()
```

Example: Full

```
import graphistry
tg = graphistry.tigergraph()
tg.gsql("""
INTERPRET QUERY () FOR GRAPH Storage {

    OrAccum<BOOL> @@stop;
    ListAccum<EDGE> @@edgeList;
    SetAccum<vertex> @@set;

    @@set += to_vertex("61921", "Pool");

    Start = @@set;

    while Start.size() > 0 and @@stop == false do

    Start = select t from Start:s-(:e)-:t
    where e.goUpper == TRUE
    accum @@edgeList += e
    having t.type != "Service";
    end;

    print @@my_edge_list;
}
""", {'edges': 'my_edge_list'}).plot()
```

static gsql_endpoint(*self*, *method_name*, *args={}*, *bindings=None*, *db=None*, *dry_run=False*)

Invoke Tigergraph stored procedure at a user-definend endpoint and return transformed Plottable

Parameters

- **method_name** (*str*) – Stored procedure name
- **args** (*Optional[dict]*) – Named endpoint arguments
- **bindings** (*Optional[dict]*) – Mapping defining names of returned 'edges' and/or 'nodes', defaults to @@nodeList and @@edgeList
- **db** (*Optional[str]*) – Name of the database, defaults to value set in .tigergraph(...)

- **dry_run** (*bool*) – Return target URL without running

Returns Plotter

Return type *Plotter*

Example: Minimal

```
import graphistry
tg = graphistry.tigergraph(db='my_db')
tg.gsql_endpoint('neighbors').plot()
```

Example: Full

```
import graphistry
tg = graphistry.tigergraph()
tg.gsql_endpoint('neighbors', {'k': 2}, {'edges': 'my_edge_list'}, 'my_db
↪').plot()
```

Example: Read data

```
import graphistry
tg = graphistry.tigergraph()
out = tg.gsql_endpoint('neighbors')
(nodes_df, edges_df) = (out._nodes, out._edges)
```

static hypergraph(*raw_events*, *entity_types=None*, *opts={}*, *drop_na=True*, *drop_edge_attrs=False*, *verbose=True*, *direct=False*, *engine='pandas'*, *npartitions=None*, *chunksize=None*)

Transform a dataframe into a hypergraph.

Parameters

- **raw_events** (*pandas.DataFrame*) – Dataframe to transform (pandas or cudf).
- **entity_types** (*Optional[list]*) – Columns (strings) to turn into nodes, None signifies all
- **opts** (*dict*) – See below
- **drop_edge_attrs** (*bool*) – Whether to include each row's attributes on its edges, defaults to False (include)
- **verbose** (*bool*) – Whether to print size information
- **direct** (*bool*) – Omit hypernode and instead strongly connect nodes in an event
- **engine** (*bool*) – String (pandas, cudf, ...) for engine to use
- **npartitions** (*Optional[int]*) – For distributed engines, how many coarse-grained pieces to split events into
- **chunksize** (*Optional[int]*) – For distributed engines, split events after chunksize rows

Create a graph out of the dataframe, and return the graph components as dataframes, and the renderable result Plotter. Hypergraphs reveal relationships between rows and between column values. This transform is useful for lists of events, samples, relationships, and other structured high-dimensional data.

Specify local compute engine by passing *engine='pandas'*, 'cudf', 'dask', 'dask_cudf' (default: 'pandas'). If events are not in that engine's format, they will be converted into it.

The transform creates a node for every unique value in the entity_types columns (default: all columns). If direct=False (default), every row is also turned into a node. Edges are added to connect every table cell to its originating row's node, or if direct=True, to the other nodes from the same row. Nodes are given the attribute 'type' corresponding to the originating column name, or in the case of a row, 'EventID'. Options further control the transform, such column category definitions for controlling whether values reocurring in different columns should be treated as one node, or whether to only draw edges between certain column type pairs.

Consider a list of events. Each row represents a distinct event, and each column some metadata about an event. If multiple events have common metadata, they will be transitively connected through those metadata values. The layout algorithm will try to cluster the events together. Conversely, if an event has unique metadata, the unique metadata will turn into nodes that only have connections to the event node, and the clustering algorithm will cause them to form a ring around the event node.

Best practice is to set EVENTID to a row's unique ID, SKIP to all non-categorical columns (or entity_types to all categorical columns), and CATEGORY to group columns with the same kinds of values.

To prevent creating nodes for null values, set drop_na=True. Some dataframe engines may have undesirable null handling, and recommend replacing None values with np.nan .

The optional `opts={...}` configuration options are:

- 'EVENTID': Column name to inspect for a row ID. By default, uses the row index.
- 'CATEGORIES': Dictionary mapping a category name to inhabiting columns. E.g., {'IP': ['srcAddress', 'dstAddress']}. If the same IP appears in both columns, this makes the transform generate one node for it, instead of one for each column.
- 'DELIM': When creating node IDs, defines the separator used between the column name and node value
- 'SKIP': List of column names to not turn into nodes. For example, dates and numbers are often skipped.
- 'EDGES': For direct=True, instead of making all edges, pick column pairs. E.g., {'a': ['b', 'd'], 'd': ['d']} creates edges between columns a->b and a->d, and self-edges d->d.

Returns {'entities': DF, 'events': DF, 'edges': DF, 'nodes': DF, 'graph': Plotter}

Return type dict

Example: Connect user<-row->boss

```
import graphistry
users_df = pd.DataFrame({'user': ['a','b','x'], 'boss': ['x', 'x', 'y
↪']})
h = graphistry.hypergraph(users_df)
g = h['graph'].plot()
```

Example: Connect user->boss

```
import graphistry
users_df = pd.DataFrame({'user': ['a','b','x'], 'boss': ['x', 'x', 'y
↪']})
h = graphistry.hypergraph(users_df, direct=True)
g = h['graph'].plot()
```

Example: Connect user<->boss

```
import graphistry
users_df = pd.DataFrame({'user': ['a','b','x'], 'boss': ['x', 'x', 'y
↪']})
h = graphistry.hypergraph(users_df, direct=True, opts={'EDGES': {'user
↪': ['boss'], 'boss': ['user']}})
g = h['graph'].plot()
```

Example: Only consider some columns for nodes

```
import graphistry
users_df = pd.DataFrame({'user': ['a','b','x'], 'boss': ['x', 'x', 'y
↪']})
h = graphistry.hypergraph(users_df, entity_types=['boss'])
g = h['graph'].plot()
```

Example: Collapse matching user::<id> and boss::<id> nodes into one person::<id> node

```
import graphistry
users_df = pd.DataFrame({'user': ['a','b','x'], 'boss': ['x', 'x', 'y
↪']})
h = graphistry.hypergraph(users_df, opts={'CATEGORIES': {'person': [
↪'user', 'boss']}})
g = h['graph'].plot()
```

Example: Use cudf engine instead of pandas

```
import cudf, graphistry
users_gdf = cudf.DataFrame({'user': ['a','b','x'], 'boss': ['x', 'x',
↪'y']})
h = graphistry.hypergraph(users_gdf, engine='cudf')
g = h['graph'].plot()
```

Parameters

- **entity_types** (Optional[List[str]]) –
- **opts** (dict) –
- **drop_na** (bool) –
- **drop_edge_attrs** (bool) –
- **verbose** (bool) –
- **direct** (bool) –
- **engine** (str) –
- **npartitions** (Optional[int]) –
- **chunksize** (Optional[int]) –

static login(*username*, *password*, *fail_silent=False*)

Authenticate and set token for reuse (api=3). If token_refresh_ms (default: 10min), auto-refreshes token. By default, must be reinvoked within 24hr.

static name(*name*)

Upload name

Parameters **name** (*str*) – Upload name

static nodes(*nodes*, *node=None*)
Specify the set of nodes and associated data.

Must include any nodes referenced in the edge list.

Parameters **nodes** – Nodes and their attributes.

Returns Plotter

Return type *Plotter*

Example

```
import graphistry

es = pd.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
g = graphistry
    .bind(source='src', destination='dst')
    .edges(es)

vs = pd.DataFrame({'v': [0,1,2], 'lbl': ['a', 'b', 'c']})
g = g.bind(node='v').nodes(vs)

g.plot()
```

Example

```
import graphistry

es = pd.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
g = graphistry.edges(es, 'src', 'dst')

vs = pd.DataFrame({'v': [0,1,2], 'lbl': ['a', 'b', 'c']})
g = g.nodes(vs, 'v)

g.plot()
```

static nodexl(*xls_or_url*, *source='default'*, *engine=None*, *verbose=False*)

Parameters

- **xls_or_url** – file/http path string to a nodexl-generated xls, or a pandas ExcelFile() object
- **source** – optionally activate binding by string name for a known nodexl data source ('twitter', 'wikimedia')
- **engine** – optionally set a pandas Excel engine
- **verbose** – optionally enable printing progress by overriding to True

static not_implemented_thunk()

static protocol(*value=None*)
Set or get the protocol ('http' or 'https'). Set automatically when using a server alias. Also set via environment variable GRAPHISTRY_PROTOCOL.

static refresh(*token=None*, *fail_silent=False*)
Use self or provided JWT token to get a fresher one. If self token, internalize upon refresh.

static register(*key=None*, *username=None*, *password=None*, *token=None*, *server=None*, *protocol=None*, *api=None*, *certificate_validation=None*, *bolt=None*, *token_refresh_ms=600000*, *store_token_creds_in_memory=None*, *client_protocol_hostname=None*)

API key registration and server selection

Changing the key effects all derived Plotter instances.

Provide one of key (api=1,2) or username/password (api=3) or token (api=3).

Parameters

- **key** (*Optional[str]*) – API key (1.0 API).
- **username** (*Optional[str]*) – Account username (2.0 API).
- **password** (*Optional[str]*) – Account password (2.0 API).
- **token** (*Optional[str]*) – Valid Account JWT token (2.0). Provide token, or username/password, but not both.
- **server** (*Optional[str]*) – URL of the visualization server.
- **certificate_validation** (*Optional[bool]*) – Override default-on check for valid TLS certificate by setting to True.
- **bolt** (*Union[dict, Any]*) – Neo4j bolt information. Optional driver or named constructor arguments for instantiating a new one.
- **protocol** (*Optional[str]*) – Protocol used to contact visualization server, defaults to "https".
- **token_refresh_ms** (*int*) – Ignored for now; JWT token auto-refreshed on plot() calls.
- **store_token_creds_in_memory** (*Optional[bool]*) – Store username/password in-memory for JWT token refreshes (Token-originated have a hard limit, so always-on requires creds somewhere)
- **client_protocol_hostname** (*Optional[str]*) – Override protocol and host shown in browser. Defaults to protocol/server or envvar GRAPHISTRY_CLIENT_PROTOCOL_HOSTNAME.

Returns None.

Return type None

Example: Standard (2.0 api by username/password)

```
import graphistry
graphistry.register(api=3, protocol='http', server='200.1.1.1', username=
↪'person', password='pwd')
```

Example: Standard (2.0 api by token)

```
import graphistry
graphistry.register(api=3, protocol='http', server='200.1.1.1', token='abc
↪')
```

Example: Remote browser to Graphistry-provided notebook server (2.0)

```
import graphistry
graphistry.register(api=3, protocol='http', server='nginx', client_
↪protocol_hostname='https://my.site.com', token='abc')
```

(continues on next page)

(continued from previous page)

Example: Standard (1.0)

```
import graphistry
graphistry.register(api=1, key="my api key")
```

relogin()

static server(*value=None*)
Get the hostname of the server or set the server using hostname or aliases. Also set via environment variable GRAPHISTRY_HOSTNAME.

static set_bolt_driver(*driver=None*)

static settings(*height=None*, *url_params={}*, *render=None*)

static store_token_creds_in_memory(*value=None*)
Cache credentials for JWT token access. Default off due to not being safe.

static style(*bg=None*, *fg=None*, *logo=None*, *page=None*)
Creates a base plotter with some style settings.

For parameters, see `plotter.style`.

Returns Plotter

Return type *Plotter*

Example

```
import graphistry
graphistry.style(bg={'color': 'black'})
```

static tigergraph(*protocol='http'*, *server='localhost'*, *web_port=14240*, *api_port=9000*, *db=None*, *user='tigergraph'*, *pwd='tigergraph'*, *verbose=False*)
Register Tigergraph connection setting defaults

Parameters

- **protocol** (*Optional[str]*) – Protocol used to contact the database.
- **server** (*Optional[str]*) – Domain of the database
- **web_port** (*Optional[int]*) –
- **api_port** (*Optional[int]*) –
- **db** (*Optional[str]*) – Name of the database
- **user** (*Optional[str]*) –
- **pwd** (*Optional[str]*) –
- **verbose** (*Optional[bool]*) – Whether to print operations

Returns Plotter

Return type *Plotter*

Example: Standard

```
import graphistry
tg = graphistry.tigergraph(protocol='https', server='acme.com', db='my_db
↪', user='alice', pwd='tigergraph2')
```

static verify_token(*token=None*, *fail_silent=False*)

Return True iff current or provided token is still valid

Return type bool

graphistry.pygraphistry.**addStyle**(*bg=None*, *fg=None*, *logo=None*, *page=None*)

Creates a base plotter with some style settings.

For parameters, see plotter.addStyle.

Returns Plotter

Return type *Plotter*

Example

```
import graphistry
graphistry.addStyle(bg={'color': 'black'})
```

graphistry.pygraphistry.**api_token**(*value=None*)

Set or get the API token. Also set via environment variable GRAPHISTRY_API_TOKEN.

graphistry.pygraphistry.**bind**(*node=None*, *source=None*, *destination=None*, *edge_title=None*, *edge_label=None*, *edge_color=None*, *edge_weight=None*, *edge_icon=None*, *edge_size=None*, *edge_opacity=None*, *edge_source_color=None*, *edge_destination_color=None*, *point_title=None*, *point_label=None*, *point_color=None*, *point_weight=None*, *point_icon=None*, *point_size=None*, *point_opacity=None*, *point_x=None*, *point_y=None*)

Create a base plotter.

Typically called at start of a program. For parameters, see plotter.bind() .

Returns Plotter

Return type *Plotter*

Example

```
import graphistry
g = graphistry.bind()
```

graphistry.pygraphistry.**bolt**(*driver=None*)

Parameters **driver** – Neo4j Driver or arguments for GraphDatabase.driver(**{...}**)

Returns Plotter w/neo4j

Call this to create a Plotter with an overridden neo4j driver.

Example

```
import graphistry
g = graphistry.bolt({ server: 'bolt://...', auth: ('<username>', '
↪<password>') })
```

```
import neo4j
import graphistry

driver = neo4j.GraphDatabase.driver(...)

g = graphistry.bolt(driver)
```

graphistry.pygraphistry.**client_protocol_hostname**(*value=None*)

Get/set the client protocol+hostname for when display urls (distinct from uploading). Also set via environment variable GRAPHISTRY_CLIENT_PROTOCOL_HOSTNAME. Defaults to hostname and no protocol (reusing environment protocol)

graphistry.pygraphistry.**cypher**(*query*, *params={}*)

Parameters

- **query** – a cypher query
- **params** – cypher query arguments

Returns Plotter with data from a cypher query. This call binds *source*, *destination*, and *node*.

Call this to immediately execute a cypher query and store the graph in the resulting Plotter.

```
import graphistry
g = graphistry.bolt({ query='MATCH (a)-[r:PAYMENT]->(b) WHERE r.USD >␣
↪7000 AND r.USD < 10000 RETURN r ORDER BY r.USD DESC', params={
↪"AccountId": 10 })
```

graphistry.pygraphistry.**description**(*description*)

Upload description

Parameters **description** (*str*) – Upload description

graphistry.pygraphistry.**edges**(*edges*, *source=None*, *destination=None*)

Specify edge list data and associated edge attribute values.

Parameters **edges** – Edges and their attributes (Pandas dataframe, NetworkX graph, or IGraph graph)

Returns Plotter

Return type *Plotter*

Example

```
import graphistry
df = pd.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
graphistry
    .bind(source='src', destination='dst')
    .edges(df)
    .plot()
```

Example

```
import graphistry
df = pd.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
graphistry
    .edges(df, 'src', 'dst')
    .plot()
```

graphistry.pygraphistry.**encode_edge_badge**(*column*, *position='TopRight'*, *categorical_mapping=None*, *continuous_binning=None*, *default_mapping=None*, *comparator=None*, *color=None*, *bg=None*, *fg=None*, *for_current=False*, *for_default=True*, *as_text=None*, *blend_mode=None*, *style=None*, *border=None*, *shape=None*)

graphistry.pygraphistry.**encode_edge_color**(*column*, *palette=None*, *as_categorical=None*, *as_continuous=None*, *categorical_mapping=None*, *default_mapping=None*, *for_default=True*, *for_current=False*)

Set edge color with more control than bind()

Parameters

- **column** (*str*) – Data column name
- **palette** (*Optional[list]*) – Optional list of color-like strings. Ex: [“black, “#FF0”, “rgb(255,255,255)”]. Used as a gradient for continuous and round-robin for categorical.
- **as_categorical** (*Optional[bool]*) – Interpret column values as categorical. Ex: Uses palette via round-robin when more values than palette entries.
- **as_continuous** (*Optional[bool]*) – Interpret column values as continuous. Ex: Uses palette for an interpolation gradient when more values than palette entries.
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to color-like strings. Ex: {“car”: “red”, “truck”: #000”}
- **default_mapping** (*Optional[str]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=”gray”.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.

Returns Plotter

Return type *Plotter*

Example: See encode_point_color

graphistry.pygraphistry.**encode_edge_icon**(*column*, *categorical_mapping=None*, *continuous_binning=None*, *default_mapping=None*, *comparator=None*, *for_default=True*, *for_current=False*, *as_text=False*, *blend_mode=None*, *style=None*, *border=None*, *shape=None*)

Set edge icon with more control than bind(). Values from Font Awesome 4 such as “laptop”: https://fontawesome.com/v4.7.0/icons/

Parameters

- **column** (*str*) – Data column name
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to icon name strings. Ex: {“toyota”: ‘car’, “ford”: ‘truck’}

- **default_mapping** (*Optional[Union[int,float]]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=50.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.
- **as_text** (*Optional[bool]*) – Values should instead be treated as raw strings, instead of icons and images. (Default False.)
- **blend_mode** (*Optional[str]*) – CSS blend mode
- **style** (*Optional[dict]*) – CSS filter properties - opacity, saturation, luminosity, grayscale, and more
- **border** (*Optional[dict]*) – Border properties - 'width', 'color', and 'storke'

Returns Plotter

Return type *Plotter*

Example: Set a string column of icons for the edge icons, same as bind(edge_icon='my_column')

```
g2a = g.encode_edge_icon('my_icons_column')
```

Example: Map specific values to specific icons, including with a default

```
g2a = g.encode_edge_icon('brands', categorical_mapping={'toyota': 'car', 'ford
↪': 'truck'})
g2b = g.encode_edge_icon('brands', categorical_mapping={'toyota': 'car', 'ford
↪': 'truck'}, default_mapping='question')
```

Example: Map countries to abbreviations

```
g2a = g.encode_edge_icon('country_abbrev', as_text=True)
g2b = g.encode_edge_icon('country', categorical_mapping={'England': 'UK',
↪'America': 'US'}, default_mapping='')
```

Example: Border

```
g2b = g.encode_edge_icon('country', border={'width': 3, color: 'black',
↪'stroke': 'dashed'}, 'categorical_mapping={'England': 'UK', 'America': 'US'}
↪)
```

graphistry.pygraphistry.**encode_point_badge**(*column*, *position='TopRight'*, *categorical_mapping=None*, *continuous_binning=None*, *default_mapping=None*, *comparator=None*, *color=None*, *bg=None*, *fg=None*, *for_current=False*, *for_default=True*, *as_text=None*, *blend_mode=None*, *style=None*, *border=None*, *shape=None*)

graphistry.pygraphistry.**encode_point_color**(*column*, *palette=None*, *as_categorical=None*, *as_continuous=None*, *categorical_mapping=None*, *default_mapping=None*, *for_default=True*, *for_current=False*)

Set point color with more control than bind()

Parameters

- **column** (*str*) – Data column name
- **palette** (*Optional[list]*) – Optional list of color-like strings. Ex: [“black, “#FF0”, “rgb(255,255,255)”]. Used as a gradient for continuous and round-robin for categorical.
- **as_categorical** (*Optional[bool]*) – Interpret column values as categorical. Ex: Uses palette via round-robin when more values than palette entries.
- **as_continuous** (*Optional[bool]*) – Interpret column values as continuous. Ex: Uses palette for an interpolation gradient when more values than palette entries.
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to color-like strings. Ex: {“car”: “red”, “truck”: #000”}
- **default_mapping** (*Optional[str]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=”gray”.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.

Returns Plotter

Return type *Plotter*

Example: Set a palette-valued column for the color, same as bind(point_color='my_column')

```
g2a = g.encode_point_color('my_int32_palette_column')
g2b = g.encode_point_color('my_int64_rgb_column')
```

Example: Set a cold-to-hot gradient of along the spectrum blue, yellow, red

```
g2 = g.encode_point_color('my_numeric_col', palette=["blue", "yellow", "red"],
↪ as_continuous=True)
```

Example: Round-robin sample from 5 colors in hex format

```
g2 = g.encode_point_color('my_distinctly_valued_col', palette=["#000", "#00F",
↪ "#0F0", "#0FF", "#FFF"], as_categorical=True)
```

Example: Map specific values to specific colors, including with a default

```
g2a = g.encode_point_color('brands', categorical_mapping={'toyota': 'red',
↪'ford': 'blue'})
g2a = g.encode_point_color('brands', categorical_mapping={'toyota': 'red',
↪'ford': 'blue'}, default_mapping='gray')
```

graphistry.pygraphistry.**encode_point_icon**(*column*, *categorical_mapping=None*, *continuous_binning=None*, *default_mapping=None*, *comparator=None*, *for_default=True*, *for_current=False*, *as_text=False*, *blend_mode=None*, *style=None*, *border=None*, *shape=None*)

Set node icon with more control than bind(). Values from Font Awesome 4 such as “laptop”: https://fontawesome.com/v4.7.0/icons/

Parameters

- **column** (*str*) – Data column name
- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to icon name strings. Ex: {"toyota": 'car', "ford": 'truck'}
- **default_mapping** (*Optional[Union[int,float]]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=50.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.
- **as_text** (*Optional[bool]*) – Values should instead be treated as raw strings, instead of icons and images. (Default False.)
- **blend_mode** (*Optional[str]*) – CSS blend mode
- **style** (*Optional[dict]*) – CSS filter properties - opacity, saturation, luminosity, grayscale, and more
- **border** (*Optional[dict]*) – Border properties - 'width', 'color', and 'storke'

Returns Plotter

Return type *Plotter*

Example: Set a string column of icons for the point icons, same as bind(point_icon='my_column')

```
g2a = g.encode_point_icon('my_icons_column')
```

Example: Map specific values to specific icons, including with a default

```
g2a = g.encode_point_icon('brands', categorical_mapping={'toyota': 'car',
↪'ford': 'truck'})
g2b = g.encode_point_icon('brands', categorical_mapping={'toyota': 'car',
↪'ford': 'truck'}, default_mapping='question')
```

Example: Map countries to abbreviations

```
g2b = g.encode_point_icon('country_abbrev', as_text=True)
g2b = g.encode_point_icon('country', as_text=True, categorical_mapping={
↪'England': 'UK', 'America': 'US'}, default_mapping='')
```

Example: Border

```
g2b = g.encode_point_icon('country', border={'width': 3, color: 'black',
↪'stroke': 'dashed'}, 'categorical_mapping={'England': 'UK', 'America': 'US'}
↪)
```

graphistry.pygraphistry.**encode_point_size**(*column*, *categorical_mapping=None*, *default_mapping=None*, *for_default=True*, *for_current=False*)

Set point size with more control than bind()

Parameters

- **column** (*str*) – Data column name

- **categorical_mapping** (*Optional[dict]*) – Mapping from column values to numbers. Ex: {"car": 100, "truck": 200}
- **default_mapping** (*Optional[Union[int,float]]*) – Augment categorical_mapping with mapping for values not in categorical_mapping. Ex: default_mapping=50.
- **for_default** (*Optional[bool]*) – Use encoding for when no user override is set. Default on.
- **for_current** (*Optional[bool]*) – Use encoding as currently active. Clearing the active encoding resets it to default, which may be different. Default on.

Returns Plotter

Return type *Plotter*

Example: Set a numerically-valued column for the size, same as bind(point_size='my_column')

```
g2a = g.encode_point_size('my_numeric_column')
```

Example: Map specific values to specific colors, including with a default

```
g2a = g.encode_point_size('brands', categorical_mapping={'toyota': 100, 'ford
↪': 200})
g2b = g.encode_point_size('brands', categorical_mapping={'toyota': 100, 'ford
↪': 200}, default_mapping=50)
```

graphistry.pygraphistry.**graph**(*ig*)

graphistry.pygraphistry.**gsql**(*query*, *bindings=None*, *dry_run=False*)

Run Tigergraph query in interpreted mode and return transformed Plottable

param query Code to run

type query str

param bindings Mapping defining names of returned 'edges' and/or 'nodes', defaults to @@nodeList and @@edgeList

type bindings Optional[dict]

param dry_run Return target URL without running

type dry_run bool

returns Plotter

rtype Plotter

Example: Minimal

```
import graphistry
tg = graphistry.tigergraph()
tg.gsql("""
INTERPRET QUERY () FOR GRAPH Storage {

    OrAccum<BOOL> @@stop;
    ListAccum<EDGE> @@edgeList;
    SetAccum<vertex> @@set;
```

(continues on next page)

(continued from previous page)

```
    @@set += to_vertex("61921", "Pool");

    Start = @@set;

    while Start.size() > 0 and @@stop == false do

    Start = select t from Start:s-(:e)-:t
    where e.goUpper == TRUE
    accum @@edgeList += e
    having t.type != "Service";
    end;

    print @@edgeList;
}
""").plot()
```

Example: Full

```
import graphistry
tg = graphistry.tigergraph()
tg.gsql("""
INTERPRET QUERY () FOR GRAPH Storage {

    OrAccum<BOOL> @@stop;
    ListAccum<EDGE> @@edgeList;
    SetAccum<vertex> @@set;

    @@set += to_vertex("61921", "Pool");

    Start = @@set;

    while Start.size() > 0 and @@stop == false do

    Start = select t from Start:s-(:e)-:t
    where e.goUpper == TRUE
    accum @@edgeList += e
    having t.type != "Service";
    end;

    print @@my_edge_list;
}
""", {'edges': 'my_edge_list'}).plot()
```

graphistry.pygraphistry.**gsql_endpoint**(*self*, *method_name*, *args={}*, *bindings=None*, *db=None*, *dry_run=False*)

Invoke Tigergraph stored procedure at a user-definend endpoint and return transformed Plottable

Parameters

- **method_name** (*str*) – Stored procedure name
- **args** (*Optional[dict]*) – Named endpoint arguments
- **bindings** (*Optional[dict]*) – Mapping defining names of returned 'edges' and/or 'nodes', defaults to @@nodeList and @@edgeList
- **db** (*Optional[str]*) – Name of the database, defaults to value set in .tigergraph(...)

- **dry_run** (*bool*) – Return target URL without running

Returns Plotter

Return type *Plotter*

Example: Minimal

```
import graphistry
tg = graphistry.tigergraph(db='my_db')
tg.gsql_endpoint('neighbors').plot()
```

Example: Full

```
import graphistry
tg = graphistry.tigergraph()
tg.gsql_endpoint('neighbors', {'k': 2}, {'edges': 'my_edge_list'}, 'my_db').
↪plot()
```

Example: Read data

```
import graphistry
tg = graphistry.tigergraph()
out = tg.gsql_endpoint('neighbors')
(nodes_df, edges_df) = (out._nodes, out._edges)
```

graphistry.pygraphistry.**hypergraph**(*raw_events*, *entity_types=None*, *opts={}*, *drop_na=True*, *drop_edge_attrs=False*, *verbose=True*, *direct=False*, *engine='pandas'*, *npartitions=None*, *chunksize=None*)

Transform a dataframe into a hypergraph.

Parameters

- **raw_events** (*pandas.DataFrame*) – Dataframe to transform (pandas or cudf).
- **entity_types** (*Optional[list]*) – Columns (strings) to turn into nodes, None signifies all
- **opts** (*dict*) – See below
- **drop_edge_attrs** (*bool*) – Whether to include each row's attributes on its edges, defaults to False (include)
- **verbose** (*bool*) – Whether to print size information
- **direct** (*bool*) – Omit hypernode and instead strongly connect nodes in an event
- **engine** (*bool*) – String (pandas, cudf, ...) for engine to use
- **npartitions** (*Optional[int]*) – For distributed engines, how many coarse-grained pieces to split events into
- **chunksize** (*Optional[int]*) – For distributed engines, split events after chunksize rows

Create a graph out of the dataframe, and return the graph components as dataframes, and the renderable result Plotter. Hypergraphs reveal relationships between rows and between column values. This transform is useful for lists of events, samples, relationships, and other structured high-dimensional data.

Specify local compute engine by passing *engine='pandas'*, 'cudf', 'dask', 'dask_cudf' (default: 'pandas'). If events are not in that engine's format, they will be converted into it.

The transform creates a node for every unique value in the entity_types columns (default: all columns). If direct=False (default), every row is also turned into a node. Edges are added to connect every table cell to its originating row's node, or if direct=True, to the other nodes from the same row. Nodes are given the attribute 'type' corresponding to the originating column name, or in the case of a row, 'EventID'. Options further control the transform, such column category definitions for controlling whether values reocurring in different columns should be treated as one node, or whether to only draw edges between certain column type pairs.

Consider a list of events. Each row represents a distinct event, and each column some metadata about an event. If multiple events have common metadata, they will be transitively connected through those metadata values. The layout algorithm will try to cluster the events together. Conversely, if an event has unique metadata, the unique metadata will turn into nodes that only have connections to the event node, and the clustering algorithm will cause them to form a ring around the event node.

Best practice is to set EVENTID to a row's unique ID, SKIP to all non-categorical columns (or entity_types to all categorical columns), and CATEGORY to group columns with the same kinds of values.

To prevent creating nodes for null values, set drop_na=True. Some dataframe engines may have undesirable null handling, and recommend replacing None values with np.nan .

The optional `opts={...}` configuration options are:

- 'EVENTID': Column name to inspect for a row ID. By default, uses the row index.
- 'CATEGORIES': Dictionary mapping a category name to inhabiting columns. E.g., {'IP': ['srcAddress', 'dstAddress']}. If the same IP appears in both columns, this makes the transform generate one node for it, instead of one for each column.
- 'DELIM': When creating node IDs, defines the separator used between the column name and node value
- 'SKIP': List of column names to not turn into nodes. For example, dates and numbers are often skipped.
- 'EDGES': For direct=True, instead of making all edges, pick column pairs. E.g., {'a': ['b', 'd'], 'd': ['d']} creates edges between columns a->b and a->d, and self-edges d->d.

Returns {'entities': DF, 'events': DF, 'edges': DF, 'nodes': DF, 'graph': Plotter}

Return type dict

Example: Connect user<-row->boss

```
import graphistry
users_df = pd.DataFrame({'user': ['a','b','x'], 'boss': ['x', 'x', 'y']})
h = graphistry.hypergraph(users_df)
g = h['graph'].plot()
```

Example: Connect user->boss

```
import graphistry
users_df = pd.DataFrame({'user': ['a','b','x'], 'boss': ['x', 'x', 'y']})
h = graphistry.hypergraph(users_df, direct=True)
g = h['graph'].plot()
```

Example: Connect user<->boss

```
import graphistry
users_df = pd.DataFrame({'user': ['a','b','x'], 'boss': ['x', 'x', 'y']})
h = graphistry.hypergraph(users_df, direct=True, opts={'EDGES': {'user': [
↪'boss'], 'boss': ['user']}})
g = h['graph'].plot()
```

Example: Only consider some columns for nodes

```
import graphistry
users_df = pd.DataFrame({'user': ['a','b','x'], 'boss': ['x', 'x', 'y']})
h = graphistry.hypergraph(users_df, entity_types=['boss'])
g = h['graph'].plot()
```

Example: Collapse matching user::<id> and boss::<id> nodes into one person::<id> node

```
import graphistry
users_df = pd.DataFrame({'user': ['a','b','x'], 'boss': ['x', 'x', 'y']})
h = graphistry.hypergraph(users_df, opts={'CATEGORIES': {'person': ['user
↪', 'boss']}})
g = h['graph'].plot()
```

Example: Use cudf engine instead of pandas

```
import cudf, graphistry
users_gdf = cudf.DataFrame({'user': ['a','b','x'], 'boss': ['x', 'x', 'y
↪']})
h = graphistry.hypergraph(users_gdf, engine='cudf')
g = h['graph'].plot()
```

Parameters

- **entity_types** (Optional[List[str]]) –
- **opts** (dict) –
- **drop_na** (bool) –
- **drop_edge_attrs** (bool) –
- **verbose** (bool) –
- **direct** (bool) –
- **engine** (str) –
- **npartitions** (Optional[int]) –
- **chunksize** (Optional[int]) –

graphistry.pygraphistry.**login**(*username*, *password*, *fail_silent=False*)

Authenticate and set token for reuse (api=3). If token_refresh_ms (default: 10min), auto-refreshes token. By default, must be reinvoked within 24hr.

graphistry.pygraphistry.**name**(*name*)

Upload name

Parameters **name** (*str*) – Upload name

graphistry.pygraphistry.**nodes**(*nodes*, *node=None*)

Specify the set of nodes and associated data.

Must include any nodes referenced in the edge list.

Parameters **nodes** – Nodes and their attributes.

Returns Plotter

Return type *Plotter*

Example

```
import graphistry

es = pd.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
g = graphistry
    .bind(source='src', destination='dst')
    .edges(es)

vs = pd.DataFrame({'v': [0,1,2], 'lbl': ['a', 'b', 'c']})
g = g.bind(node='v').nodes(vs)

g.plot()
```

Example

```
import graphistry

es = pd.DataFrame({'src': [0,1,2], 'dst': [1,2,0]})
g = graphistry.edges(es, 'src', 'dst')

vs = pd.DataFrame({'v': [0,1,2], 'lbl': ['a', 'b', 'c']})
g = g.nodes(vs, 'v)

g.plot()
```

graphistry.pygraphistry.**nodexl**(*xls_or_url*, *source='default'*, *engine=None*, *verbose=False*)

Parameters

- **xls_or_url** – file/http path string to a nodexl-generated xls, or a pandas ExcelFile() object
- **source** – optionally activate binding by string name for a known nodexl data source ('twitter', 'wikimedia')
- **engine** – optionally set a pandas Excel engine
- **verbose** – optionally enable printing progress by overriding to True

graphistry.pygraphistry.**protocol**(*value=None*)

Set or get the protocol ('http' or 'https'). Set automatically when using a server alias. Also set via environment variable GRAPHISTRY_PROTOCOL.

graphistry.pygraphistry.**refresh**(*token=None*, *fail_silent=False*)

Use self or provided JWT token to get a fresher one. If self token, internalize upon refresh.

graphistry.pygraphistry.**register**(*key=None*, *username=None*, *password=None*, *token=None*, *server=None*, *protocol=None*, *api=None*, *certificate_validation=None*, *bolt=None*, *token_refresh_ms=600000*, *store_token_creds_in_memory=None*, *client_protocol_hostname=None*)

API key registration and server selection

Changing the key effects all derived Plotter instances.

Provide one of key (api=1,2) or username/password (api=3) or token (api=3).

Parameters

- **key** (*Optional[str]*) – API key (1.0 API).

- **username** (*Optional[str]*) – Account username (2.0 API).
- **password** (*Optional[str]*) – Account password (2.0 API).
- **token** (*Optional[str]*) – Valid Account JWT token (2.0). Provide token, or username/password, but not both.
- **server** (*Optional[str]*) – URL of the visualization server.
- **certificate_validation** (*Optional[bool]*) – Override default-on check for valid TLS certificate by setting to True.
- **bolt** (*Union[dict, Any]*) – Neo4j bolt information. Optional driver or named constructor arguments for instantiating a new one.
- **protocol** (*Optional[str]*) – Protocol used to contact visualization server, defaults to "https".
- **token_refresh_ms** (*int*) – Ignored for now; JWT token auto-refreshed on plot() calls.
- **store_token_creds_in_memory** (*Optional[bool]*) – Store username/password in-memory for JWT token refreshes (Token-originated have a hard limit, so always-on requires creds somewhere)
- **client_protocol_hostname** (*Optional[str]*) – Override protocol and host shown in browser. Defaults to protocol/server or envvar GRAPHISTRY_CLIENT_PROTOCOL_HOSTNAME.

Returns None.

Return type None

Example: Standard (2.0 api by username/password)

```
import graphistry
graphistry.register(api=3, protocol='http', server='200.1.1.1', username=
↪'person', password='pwd')
```

Example: Standard (2.0 api by token)

```
import graphistry
graphistry.register(api=3, protocol='http', server='200.1.1.1', token='abc')
```

Example: Remote browser to Graphistry-provided notebook server (2.0)

```
import graphistry
graphistry.register(api=3, protocol='http', server='nginx', client_protocol_
↪hostname='https://my.site.com', token='abc')
```

Example: Standard (1.0)

```
import graphistry
graphistry.register(api=1, key="my api key")
```

graphistry.pygraphistry.**server**(*value=None*)

Get the hostname of the server or set the server using hostname or aliases. Also set via environment variable GRAPHISTRY_HOSTNAME.

graphistry.pygraphistry.**settings**(*height=None*, *url_params={}*, *render=None*)

graphistry.pygraphistry.**store_token_creds_in_memory**(*value=None*)

Cache credentials for JWT token access. Default off due to not being safe.

graphistry.pygraphistry.**style**(*bg=None*, *fg=None*, *logo=None*, *page=None*)

Creates a base plotter with some style settings.

For parameters, see `plotter.style`.

Returns Plotter

Return type *Plotter*

Example

```
import graphistry
graphistry.style(bg={'color': 'black'})
```

graphistry.pygraphistry.**tigergraph**(*protocol='http'*, *server='localhost'*, *web_port=14240*, *api_port=9000*, *db=None*, *user='tigergraph'*, *pwd='tigergraph'*, *verbose=False*)

Register Tigergraph connection setting defaults

Parameters

- **protocol** (*Optional[str]*) – Protocol used to contact the database.
- **server** (*Optional[str]*) – Domain of the database
- **web_port** (*Optional[int]*) –
- **api_port** (*Optional[int]*) –
- **db** (*Optional[str]*) – Name of the database
- **user** (*Optional[str]*) –
- **pwd** (*Optional[str]*) –
- **verbose** (*Optional[bool]*) – Whether to print operations

Returns Plotter

Return type *Plotter*

Example: Standard

```
import graphistry
tg = graphistry.tigergraph(protocol='https', server='acme.com', db='my_db',␣
↪user='alice', pwd='tigergraph2')
```

graphistry.pygraphistry.**verify_token**(*token=None*, *fail_silent=False*)

Return True iff current or provided token is still valid

Return type bool

1.4 graphistry.arrow_uploader module

class graphistry.arrow_uploader.**ArrowUploader**(*server_base_path='http://nginx'*, *view_base_path='http://localhost'*, *name=None*, *description=None*, *edges=None*, *nodes=None*, *node_encodings=None*, *edge_encodings=None*, *token=None*, *dataset_id=None*, *metadata=None*, *certificate_validation=True*)

Bases: object

arrow_to_buffer(*table*)

Parameters **table** (Table) –

property certificate_validation

create_dataset(*json*)

property dataset_id

Return type str

property description

Return type str

property edge_encodings

property edges

Return type Table

g_to_edge_bindings(*g*)

g_to_edge_encodings(*g*)

g_to_node_bindings(*g*)

g_to_node_encodings(*g*)

login(*username*, *password*)

maybe_bindings(*g*, *bindings*, *base={}*)

property metadata

property name

Return type str

property node_encodings

property nodes

Return type Table

post(*as_files=True*, *memoize=True*)

as_files deprecation plan: Graphistry 2.34: Introduced Graphistry 2.35: Does nothing (runtime warning); all uploads are Files Graphistry 2.36: Remove flag

Parameters

- **as_files** (bool) –

- **memoize** (bool) –

post_arrow(*arr*, *graph_type*, *opts=''*)

Parameters

- **arr** (Table) –
- **graph_type** (str) –
- **opts** (str) –

post_arrow_generic(*sub_path*, *tok*, *arr*, *opts=''*)

Parameters

- **sub_path** (str) –
- **tok** (str) –
- **arr** (Table) –

Return type Response

post_edges_arrow(*arr=None*, *opts=''*)

post_edges_file(*file_path*, *file_type='csv'*)

post_file(*file_path*, *graph_type='edges'*, *file_type='csv'*)

post_g(*g*, *name=None*, *description=None*)

post_nodes_arrow(*arr=None*, *opts=''*)

post_nodes_file(*file_path*, *file_type='csv'*)

refresh(*token=None*)

property server_base_path

Return type str

property token

Return type str

verify(*token=None*)

Return type bool

property view_base_path

Return type str

1.5 graphistry.ArrowFileUploader module

class graphistry.ArrowFileUploader.**ArrowFileUploader**(*uploader*)

Bases: object

Implement file API with focus on Arrow support

Memoization in this class is based on reference equality, while plotter is based on hash. That means the plotter resolves different-identity value matches, so by the time ArrowFileUploader compares, identities are unified for faster reference-based checks.

Example: Upload files with per-session memoization uploader : ArrowUploader arr : pa.Table afu = ArrowFileUploader(uploader)

file1_id = afu.create_and_post_file(arr)[0] file2_id = afu.create_and_post_file(arr)[0]

assert file1_id == file2_id # memoizes by default (memory-safe: weak refs)

Example: Explicitly create a file and upload data for it uploader : ArrowUploader arr : pa.Table afu = ArrowFileUploader(uploader)

file1_id = afu.create_file() afu.post_arrow(arr, file_id)

file2_id = afu.create_file() afu.post_arrow(arr, file_id)

assert file1_id != file2_id

create_and_post_file(*arr*, *file_id=None*, *file_opts={}*, *upload_url_opts='erase=true'*, *memoize=True*)

Create file and upload data for it.

Default upload_url_opts='erase=true' throws exceptions on parse errors and deletes upload.

Default memoize=True skips uploading 'arr' when previously uploaded in current session

See File REST API for file_opts (file create) and upload_url_opts (file upload)

Parameters

- **arr** (Table) –
- **file_id** (Optional[str]) –
- **file_opts** (dict) –
- **upload_url_opts** (str) –
- **memoize** (bool) –

Return type Tuple[str, dict]

create_file(*file_opts={}*)

Creates File and returns file_id str.

Defauls:

- file_type: 'arrow'

See File REST API for file_opts

Parameters **file_opts** (dict) –

Return type str

post_arrow(*arr*, *file_id*, *url_opts='erase=true'*)

Upload new data to existing file id

Default url_opts='erase=true' throws exceptions on parse errors and deletes upload.

See File REST API for url_opts (file upload)

Parameters

- **arr** (Table) –
- **file_id** (str) –
- **url_opts** (str) –

Return type dict

uploader: Any = None

graphistry.ArrowFileUploader.**DF_TO_FILE_ID_CACHE: weakref.WeakKeyDictionary = <WeakKeyDict**

NOTE: Will switch to pa.Table -> ... when RAPIDS upgrades from pyarrow, which adds weakref support

class graphistry.ArrowFileUploader.**MemoizedFileUpload**(*file_id*, *output*)
Bases: object

Parameters

- **file_id**(str) –
- **output**(dict) –

file_id: str

output: dict

class graphistry.ArrowFileUploader.**WrappedTable**(*arr*)
Bases: object

Parameters **arr**(Table) –

arr: pyarrow.lib.Table

graphistry.ArrowFileUploader.**cache_arr**(*arr*)
Hold reference to most recent memoization entries Hack until RAPIDS supports Arrow 2.0, when pa.Table becomes weakly referenceable

CHAPTER

TWO

DOC

2.1 versioneer module

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

A

addStyle() (*graphistry.plotter.Plotter method*), 3
addStyle() (*graphistry.pygraphistry.PyGraphistry static method*), 17
addStyle() (*in module graphistry.pygraphistry*), 31
api_key() (*graphistry.pygraphistry.PyGraphistry static method*), 17
api_token() (*graphistry.pygraphistry.PyGraphistry static method*), 17
api_token() (*in module graphistry.pygraphistry*), 31
api_token_refresh_ms() (*graphistry.pygraphistry.PyGraphistry static method*), 17
api_version() (*graphistry.pygraphistry.PyGraphistry static method*), 17
arr (*graphistry.ArrowFileUploader.WrappedTable attribute*), 48
arrow_to_buffer() (*graphistry.arrow_uploader.ArrowUploader method*), 45
ArrowFileUploader (*class in graphistry.ArrowFileUploader*), 46
ArrowUploader (*class in graphistry.arrow_uploader*), 45
authenticate() (*graphistry.pygraphistry.PyGraphistry static method*), 17

B

bind() (*graphistry.plotter.Plotter method*), 4
bind() (*graphistry.pygraphistry.PyGraphistry static method*), 17
bind() (*in module graphistry.pygraphistry*), 31
bolt() (*graphistry.plotter.Plotter method*), 6
bolt() (*graphistry.pygraphistry.PyGraphistry static method*), 17
bolt() (*in module graphistry.pygraphistry*), 31

C

cache_arr() (*in module graphistry.ArrowFileUploader*), 48
cache_coercion() (*in module graphistry.plotter*), 16
cache_coercion_helper() (*in module graphistry.plotter*), 16
certificate_validation() (*graphistry.arrow_uploader.ArrowUploader property*), 45
certificate_validation() (*graphistry.pygraphistry.PyGraphistry static method*), 18
client_protocol_hostname() (*graphistry.pygraphistry.PyGraphistry static method*), 18
client_protocol_hostname() (*in module graphistry.pygraphistry*), 32
create_and_post_file() (*graphistry.ArrowFileUploader.ArrowFileUploader method*), 47
create_dataset() (*graphistry.arrow_uploader.ArrowUploader method*), 45
create_file() (*graphistry.ArrowFileUploader.ArrowFileUploader method*), 47
cypher() (*graphistry.plotter.Plotter method*), 6
cypher() (*graphistry.pygraphistry.PyGraphistry static method*), 18
cypher() (*in module graphistry.pygraphistry*), 32

D

dataset_id() (*graphistry.arrow_uploader.ArrowUploader property*), 45
default() (*graphistry.pygraphistry.NumpyJSONEncoder method*), 16
description() (*graphistry.arrow_uploader.ArrowUploader property*), 45
description() (*graphistry.plotter.Plotter method*), 6
description() (*graphistry.pygraphistry.PyGraphistry static method*), 18
description() (*in module graphistry.pygraphistry*), 32
DF_TO_FILE_ID_CACHE (*in module graphistry.ArrowFileUploader*), 48

E

edge_encodings() (*graphistry.arrow_uploader.ArrowUploader*

property), 45
edges() (*graphistry.arrow_uploader.ArrowUploader property*), 45
edges() (*graphistry.plotter.Plotter method*), 6
edges() (*graphistry.pygraphistry.PyGraphistry static method*), 18
edges() (*in module graphistry.pygraphistry*), 32
encode_edge_badge() (*graphistry.plotter.Plotter method*), 6
encode_edge_badge() (*graphistry.pygraphistry.PyGraphistry static method*), 19
encode_edge_badge() (*in module graphistry.pygraphistry*), 32
encode_edge_color() (*graphistry.plotter.Plotter method*), 6
encode_edge_color() (*graphistry.pygraphistry.PyGraphistry static method*), 19
encode_edge_color() (*in module graphistry.pygraphistry*), 33
encode_edge_icon() (*graphistry.plotter.Plotter method*), 7
encode_edge_icon() (*graphistry.pygraphistry.PyGraphistry static method*), 19
encode_edge_icon() (*in module graphistry.pygraphistry*), 33
encode_point_badge() (*graphistry.plotter.Plotter method*), 8
encode_point_badge() (*graphistry.pygraphistry.PyGraphistry static method*), 20
encode_point_badge() (*in module graphistry.pygraphistry*), 34
encode_point_color() (*graphistry.plotter.Plotter method*), 8
encode_point_color() (*graphistry.pygraphistry.PyGraphistry static method*), 21
encode_point_color() (*in module graphistry.pygraphistry*), 34
encode_point_icon() (*graphistry.plotter.Plotter method*), 9
encode_point_icon() (*graphistry.pygraphistry.PyGraphistry static method*), 21
encode_point_icon() (*in module graphistry.pygraphistry*), 35
encode_point_size() (*graphistry.plotter.Plotter method*), 10
encode_point_size() (*graphistry.pygraphistry.PyGraphistry static method*), 22
encode_point_size() (*in module graphistry.pygraphistry*), 36

F

file_id (*graphistry.ArrowFileUploader.MemoizedFileUpload attribute*), 48

G

g_to_edge_bindings() (*graphistry.arrow_uploader.ArrowUploader method*), 45
g_to_edge_encodings() (*graphistry.arrow_uploader.ArrowUploader method*), 45
g_to_node_bindings() (*graphistry.arrow_uploader.ArrowUploader method*), 45
g_to_node_encodings() (*graphistry.arrow_uploader.ArrowUploader method*), 45
graph() (*graphistry.plotter.Plotter method*), 10
graph() (*graphistry.pygraphistry.PyGraphistry static method*), 23
graph() (*in module graphistry.pygraphistry*), 37
graphistry.arrow_uploader module, 45
graphistry.ArrowFileUploader module, 46
graphistry.plotter module, 3
graphistry.pygraphistry module, 16
gsql() (*graphistry.plotter.Plotter method*), 11
gsql() (*graphistry.pygraphistry.PyGraphistry static method*), 23
gsql() (*in module graphistry.pygraphistry*), 37
gsql_endpoint() (*graphistry.plotter.Plotter method*), 12
gsql_endpoint() (*graphistry.pygraphistry.PyGraphistry static method*), 24
gsql_endpoint() (*in module graphistry.pygraphistry*), 38

H

hypergraph() (*graphistry.pygraphistry.PyGraphistry static method*), 25
hypergraph() (*in module graphistry.pygraphistry*), 39

I

igraph2pandas() (*graphistry.plotter.Plotter method*), 12

L

login() (*graphistry.arrow_uploader.ArrowUploader method*), 45
login() (*graphistry.pygraphistry.PyGraphistry static method*), 27
login() (*in module graphistry.pygraphistry*), 41

M

maybe_bindings() (*graphistry.arrow_uploader.ArrowUploader method*), 45
MemoizedFileUpload (*class in graphistry.ArrowFileUploader*), 48
metadata() (*graphistry.arrow_uploader.ArrowUploader property*), 45
module
    graphistry.arrow_uploader, 45
    graphistry.ArrowFileUploader, 46
    graphistry.plotter, 3
    graphistry.pygraphistry, 16

N

name() (*graphistry.arrow_uploader.ArrowUploader property*), 45
name() (*graphistry.plotter.Plotter method*), 13
name() (*graphistry.pygraphistry.PyGraphistry static method*), 27
name() (*in module graphistry.pygraphistry*), 41
networkx2pandas() (*graphistry.plotter.Plotter method*), 13
networkx_checkoverlap() (*graphistry.plotter.Plotter method*), 13
node_encodings() (*graphistry.arrow_uploader.ArrowUploader property*), 45
nodes() (*graphistry.arrow_uploader.ArrowUploader property*), 45
nodes() (*graphistry.plotter.Plotter method*), 13
nodes() (*graphistry.pygraphistry.PyGraphistry static method*), 27
nodes() (*in module graphistry.pygraphistry*), 41
nodexl() (*graphistry.plotter.Plotter method*), 13
nodexl() (*graphistry.pygraphistry.PyGraphistry static method*), 28
nodexl() (*in module graphistry.pygraphistry*), 42
not_implemented_thunk() (*graphistry.pygraphistry.PyGraphistry static method*), 28
NumpyJSONEncoder (*class in graphistry.pygraphistry*), 16

O

output (*graphistry.ArrowFileUploader.MemoizedFileUpload attribute*), 48

P

pandas2igraph() (*graphistry.plotter.Plotter method*), 14
plot() (*graphistry.plotter.Plotter method*), 14
Plotter (*class in graphistry.plotter*), 3
post() (*graphistry.arrow_uploader.ArrowUploader method*), 45
post_arrow() (*graphistry.arrow_uploader.ArrowUploader method*), 46
post_arrow() (*graphistry.ArrowFileUploader.ArrowFileUploader method*), 47
post_arrow_generic() (*graphistry.arrow_uploader.ArrowUploader method*), 46
post_edges_arrow() (*graphistry.arrow_uploader.ArrowUploader method*), 46
post_edges_file() (*graphistry.arrow_uploader.ArrowUploader method*), 46
post_file() (*graphistry.arrow_uploader.ArrowUploader method*), 46
post_g() (*graphistry.arrow_uploader.ArrowUploader method*), 46
post_nodes_arrow() (*graphistry.arrow_uploader.ArrowUploader method*), 46
post_nodes_file() (*graphistry.arrow_uploader.ArrowUploader method*), 46
protocol() (*graphistry.pygraphistry.PyGraphistry static method*), 28
protocol() (*in module graphistry.pygraphistry*), 42
PyGraphistry (*class in graphistry.pygraphistry*), 17

R

refresh() (*graphistry.arrow_uploader.ArrowUploader method*), 46
refresh() (*graphistry.pygraphistry.PyGraphistry static method*), 28
refresh() (*in module graphistry.pygraphistry*), 42
register() (*graphistry.pygraphistry.PyGraphistry static method*), 28
register() (*in module graphistry.pygraphistry*), 42
relogin() (*graphistry.pygraphistry.PyGraphistry method*), 30

S

server() (*graphistry.pygraphistry.PyGraphistry static method*), 30
server() (*in module graphistry.pygraphistry*), 43
server_base_path() (*graphistry.arrow_uploader.ArrowUploader property*), 46

set_bolt_driver() (*graphistry.pygraphistry.PyGraphistry static method*), 30
settings() (*graphistry.plotter.Plotter method*), 15
settings() (*graphistry.pygraphistry.PyGraphistry static method*), 30
settings() (*in module graphistry.pygraphistry*), 43
store_token_creds_in_memory() (*graphistry.pygraphistry.PyGraphistry static method*), 30
store_token_creds_in_memory() (*in module graphistry.pygraphistry*), 43
style() (*graphistry.plotter.Plotter method*), 15
style() (*graphistry.pygraphistry.PyGraphistry static method*), 30
style() (*in module graphistry.pygraphistry*), 44

T

tigergraph() (*graphistry.plotter.Plotter method*), 15
tigergraph() (*graphistry.pygraphistry.PyGraphistry static method*), 30
tigergraph() (*in module graphistry.pygraphistry*), 44
token() (*graphistry.arrow_uploader.ArrowUploader property*), 46

U

uploader (*graphistry.ArrowFileUploader.ArrowFileUploader attribute*), 47

V

verify() (*graphistry.arrow_uploader.ArrowUploader method*), 46
verify_token() (*graphistry.pygraphistry.PyGraphistry static method*), 31
verify_token() (*in module graphistry.pygraphistry*), 44
view_base_path() (*graphistry.arrow_uploader.ArrowUploader property*), 46

W

WeakValueWrapper (*class in graphistry.plotter*), 16
WrappedTable (*class in graphistry.ArrowFileUploader*), 48